செம்மொழியில் கற்போம்

ஷெல் ஸ்கிரிப்ட்

shell script



பணியா. பிரசன்னா

askprasanna@gmail

மின்னூல் வெளியீடு :

FreeTamilEbooks.com

அட்டைப்படம், மின்னூலாக்கம் :

மீ.வேல். பிரசன்னா

udpmprasanna@gmail.com

, ,

.



:

“ ”

(Linux) .

,

.

,

.

,

.

.

,

.

,

.

,

, .

,

,

.



“ ” ,

, ,

(Devices /Gedgets)

(Operting System).

Linux

. Linux

. ,

.

.

,

(,

, ,)

.

,

.

.

,

. .

, ,

.



:



,

,

,

. . ,

,

.

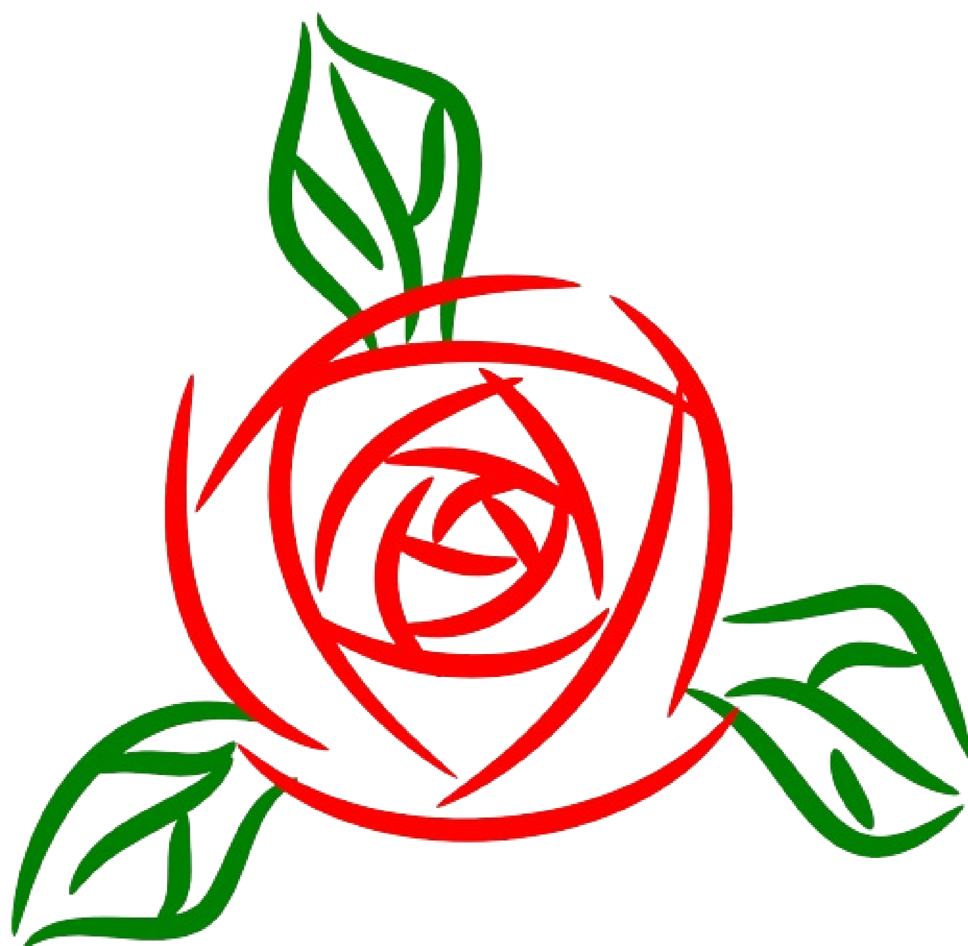
“[illegible]

[illegible]

[illegible]; [illegible]

[illegible]

[illegible] [illegible]!!”



- 1

“

”

-1

:

,

.

.

.

. .

,

,

.

.

.

?

.

.

, ++

.

.

.

.

.

(step by step execution) .

.

.

.

:

(System Administrators),
(software engineers),
(programmers), , **(Unix & Linux enthusiasts)**

.

(OS architecture),

(at least beginner level programming skills)

.

(Shebang)

,

.

, , ,

. #!

.

:

.

.

.

.

.

(

) .

1. (Bourne shell (sh))

2. (Boune Again SHell (BASH))

3. (C Shell (csh))

4. (TC Shell (tsh))

5. (Korn shell (ksh))

,

. sh ,

.

.

:

.

zsh

.

.

Criteria		sh	ksh	bash	zsh	csh	tcsh
Configurability	1	-	+	++	+++	+	++
Execution of commands	2	+	+	+	++	+	++
Completion	3	- -	+	++	+++	+	++
Line editing	4	-	+	++	++	-	++
Name substitution	5	+	+	++	++	+	++
History	6	- -	+	++	++	+	++
Redirections and pipes	7	+	+	+	++	+	+
Spelling correction	8	- -	- -	- -	+	- -	+
Prompt settings	9	+	+	+	++	+	++
Job control	10	- -	+	+	+	+	+
Execution control	11	+	+	+	+	+	+
Signal Handling	12	+	+	+	+	-	-

+++

++

+

- .

--

,

,

.

:

.

.

.

.

,

.

.

,

.

1:

```
#!/bin/bash
echo “Welcome to Linux Shell script world”
```

:

, ( )

(vi(m) editor)

. , script1.sh

. , .sh

.

,

.

#sh -x script1.sh

.

#sh -v script1.sh

.

.

.

#chmod +x script.sh

.

.

#./script.sh

. ,

.

,

, ,

.

Welcome to Linux Shell script world

. echo

.

:

?

:

, .

echo

.

batch file programming

. (filename.bat)

.

.

.

1

2 ,

,

.

. **Cygwin**

(Linux Simulation)

.

```
Windows PowerShell
PS C:\Users\jtwist> get-help about_regular_expression
TOPIC
    Regular expressions


SHORT DESCRIPTION
    Using regular expressions in Cmdlet parameters in the Windows PowerShell


LONG DESCRIPTION




    PowerShell supports a number of the regular expression characters:


        Format    Logic                               Example
        --------  -------------------------------     ------------------------
        value     Matches exact characters            "book" -match "oo"
                  anywhere in the original value
        .         Matches any single character        "copy" -match "c..y"
        [value]   Matches at least one of the         "big" -match "b[iou]g"
                  characters in the brackets
        [range]   Matches at least one of the         "and" -match "[a-e]nd"
                  characters within the range.
                  The Use of a hyphen (-) allows
                  specification of contiguous
                  character.
        [^]       Matches any character except        "and" -match "[^brt]nd"
                  those in brackets
        ^         Matches the beginning               "book" -match "^bo"
                  characters
        $         Matches the end characters          "book" -match "ok$"
        *         Matches zero or more instances      "baggy" -match "g*"
                  of the preceding character
        ?         Matches zero or one instance        "baggy" -match "g?"
                  of the preceding character
        \         Matches the character that          "Try$" -match "Try\$"
                  follows as an escaped character


    PowerShell supports the character classes available in .NET regular
    expressions
        Format    Logic                               Example
        --------  -------------------------------     ------------------------
        \p{name}  Matches any character in the        "abcd defg" -match "\p{Ll}+"
                  named character class specified
```

.

.

i.

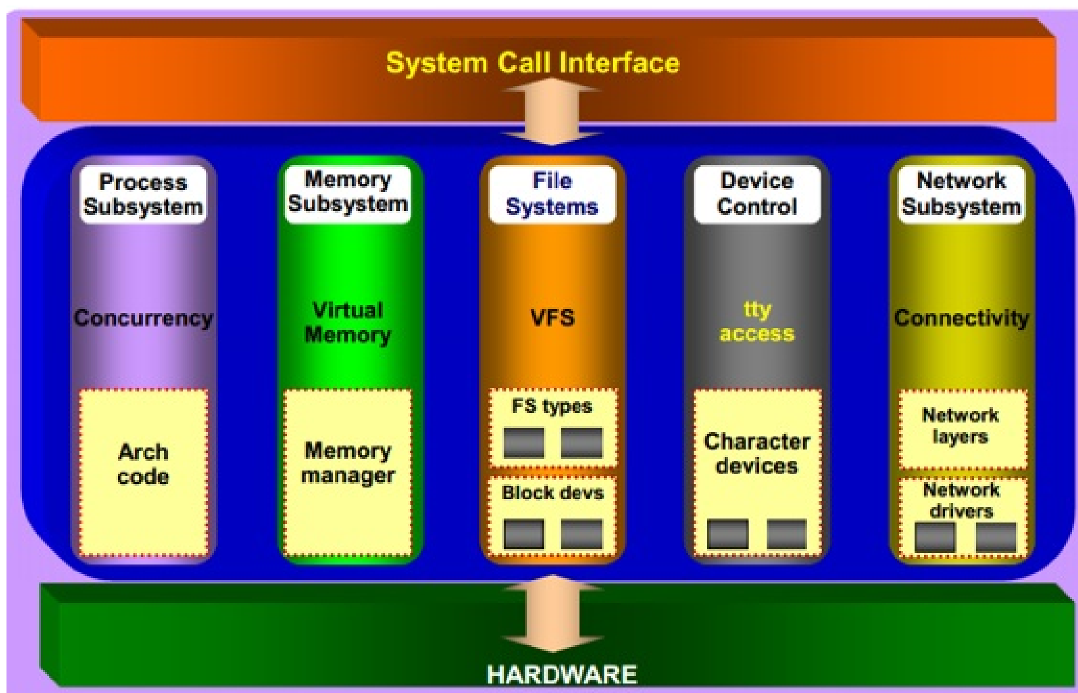
ii.

iii.

iv.

v.

.



:

– generalization

– interface

– scripts

– automation

– by default

– system

– architecture

– 2

.

i. **(process management)**

.

.

, , ,

.

i. **(memory management)**

.

,

.

RAM (Random Access Memory)

.

.

:

, **RAM**

, **RAM**

.

RAM

.

, **RAM**

.

i. **(file management)**

.

.

. **ext4, ext3, ext2, vfat, swap**

, **VFS** **Virtual File System**

. **VFS**

.

.

ls listing

.

.

fork

( ) , execute(“ls”)

. (

.)

(/ /)

(ls)

.

#ls listing

#strace ls listing

.

i. (hardware management)

,

, ,

, ,

.

,

( )

.

.

i. **(network management)**

.

1.

2.

.

.

.

.

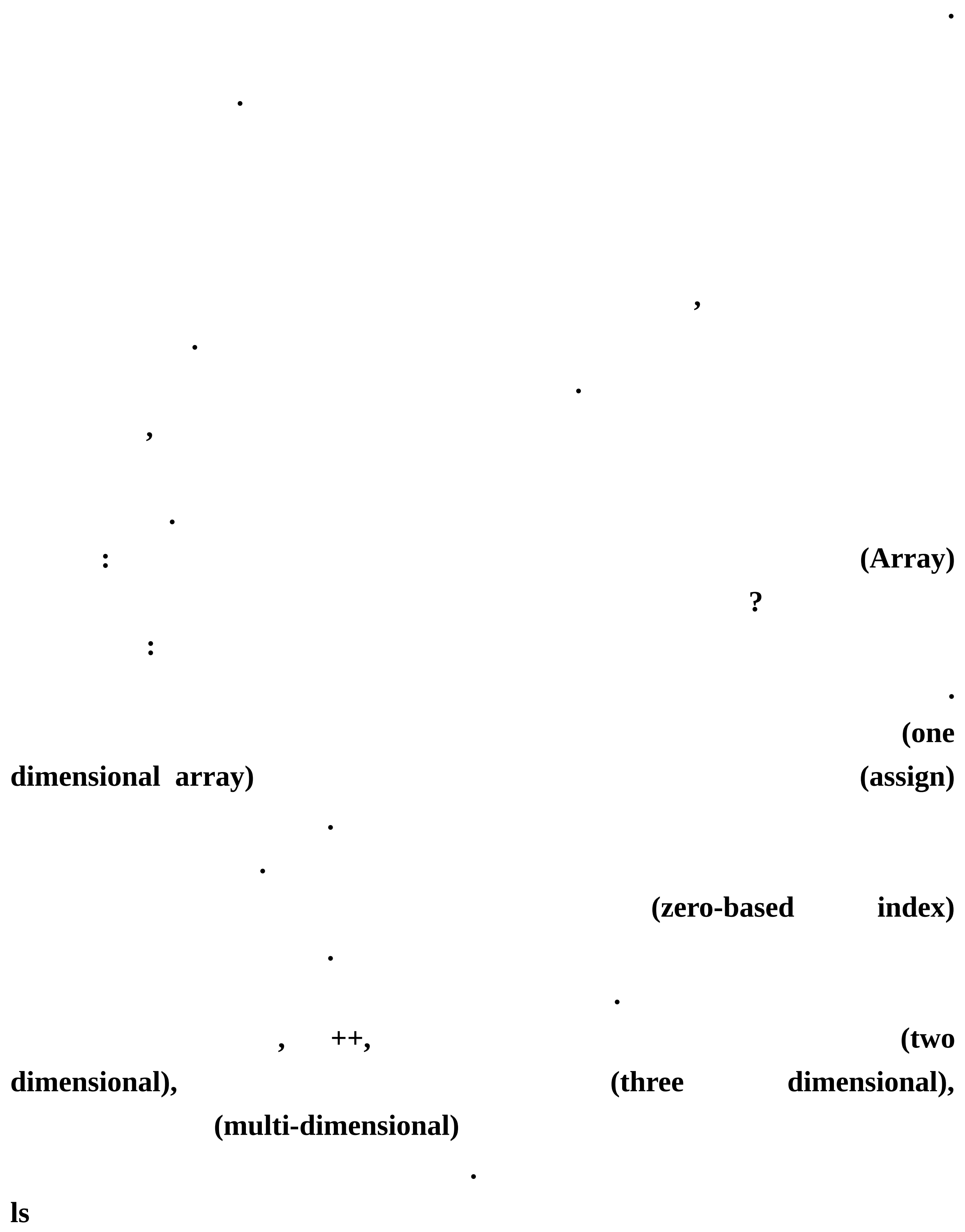
. , ++,

.

.

, , ,

.



.

.

,

.

.

,

.

: (Array)

?

:

.

(one

dimensional array) (assign)

.

.

(zero-based index)

.

.

, ++, (two

dimensional), (three dimensional),

(multi-dimensional)

.

ls

.

ls command
⇩
execute("ls")
⇩
device open in VFS
⇩
data part of Harddisk

VFS Virtual File System

.

.

:

i=5

j=6

:

one=”This is one”

two=”This is two”

three=”3” 3

.

.

2:

```
#!/bin/bash
echo “i value is $i”
echo “j value is $j”
echo “first string is $one”
echo “second string is $two”
echo “third string is $three”
```

(shellscript) ?

.

1. குறுநிரலானது பிழைகளின்றி இருத்தல் வேண்டும்.
2. நிரலானது ஏரணப் பிழையறத் தெளிவுற அமைதல் வேண்டும்.
3. நிரலானது ஒரு குறிப்பிட்ட செயலினைச் செய்தல் வேண்டும்.
4. குறுநிரலானது தேவையற்ற செயல்களைச் செய்தல் கூடாது.
5. குறுநிரல்கள் திரும்பப் பயன்படுவதற்கு ஏற்றதாக அமைதல் வேண்டும்.

3:

(system login users)

.

w **who**

.

.

#who | sort | cut -d‘ ’ -f1

who . **sort**

. **cut**

field **(column)**

. **d** **delimiter or separator**

. **f1**

(column) .

.

```
#!/bin/bash
# whos – a program which displays the login usernames of a particular system.
who | sort | cut -d‘ ’ -f1
```

. (vim

whos.sh; chmod +x whos.sh; ./whos.sh)

```
root@localhost:~/Desktop/scripts
File  Edit  View  Search  Terminal  Help
#!/bin/bash
# whos – a program which displays the login usernames of a particular system.

who | sort | cut -d' ' -f1

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"whos.sh" 5L, 121C written
```

```
root@localhost:~/Desktop/scripts
File  Edit  View  Search  Terminal  Help
[root@localhost scripts]# cat whos.sh
#!/bin/bash
# whos – a program which displays the login usernames of a particular system.

who | sort | cut -d' ' -f1

[root@localhost scripts]# chmod +x whos.sh
[root@localhost scripts]# ./whos.sh
root
root
[root@localhost scripts]#
```

□□□□□□□□□□□□□:

- Basic arithmetic operations

- Primary memory

- Secondary memory

- Virtual memory or swap memory

- Temporary file system

- Numbers, Integers, Floats, Double etc.

- Texts, Strings

- Printers

- Backup tapes

- Wired network

- Wireless network

- predefined network layers

- network device drivers

- logic

- data

- declaration

- zero

(…)

- 3 ,

.

- நிரற்பா 3

:

,

.

parent process , child process, orphan process daemon process .

.

(running or active state), (stopping state), (dead or zombie state), (sleeping state)

(interruptable sleep state), (uninterruptable sleeping state) .

(Parent Process)

.

(Child Process) .

,

(Orphan Process)

.

(Daemon Process)

.

(Zombie state) ,

,

.

.

,

.

(Stop state) .

:

,

.

Type of processes in Linux
Parent (பெற்றோர்)
Child (குழந்தை)
Orphan (இல்லார்)
Daemon (உதவியர்)

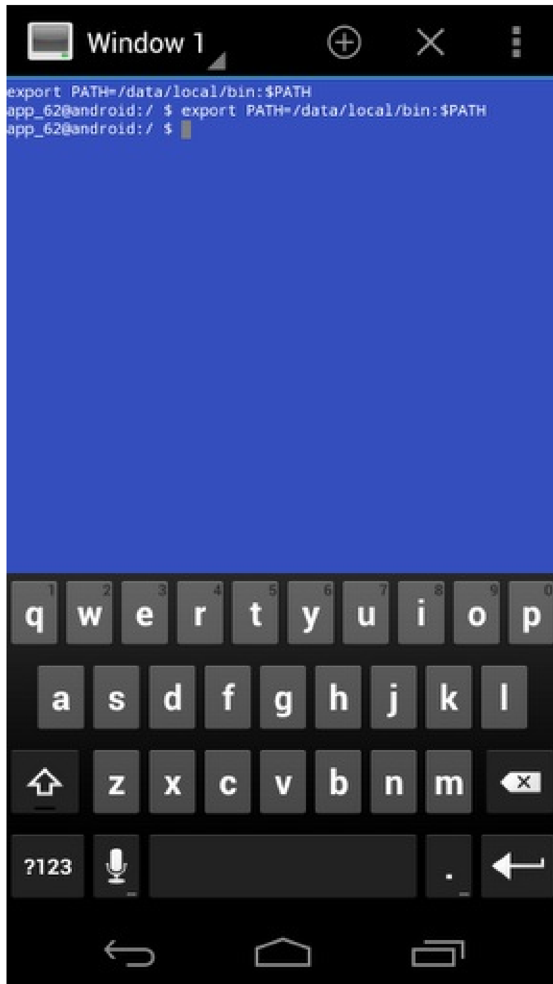
State of processes
Running (ஓட்டம்)
Sleeping (உறக்கம்)
Uninterrupted sleep (குறுக்கிடு உறக்கம்)
Interrupted sleep (குறுக்கிடா உறக்கம்)
Stopping (நிற்றல்)
Zombie (உள்ளுறை) உள்ளே உறைந்து இறந்த

uninterruptible sleep
Signal
Stopped
uninterruptible sleep
Signal
wake
Running
Exit
interruptible sleep
Zombie
interruptible sleep
Signal
Process Created

:

?

□□□□□□□: Terminal emulator □□□□□□□□ □□□□□□□□□□□ □□□□□□□□□□□□□ echo □□□□□□□ □□□□□□□□□□□ □□□□□□□□□□□. http://www.appsapk.com/android-terminal-emulator/ □□□□□ □□□□□□□□□□□□□ □□□ □□□□□□□□□□□□□. □□□□□□□□□□□□□□□□□ □□□□□□□□□□□ □□□□□□. □□□□□ □□□ □□□□□□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□□□□□. □□□□□□□□□ □□□□□□□ □□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□ □□□□□□□□□□□ root □□□□□□□□ hack □□□□□□□□□□□ □□□□□□□□□□.



Complete Linux Installer http://www.appsapk.com/complete-linux-installer/ [illegible]



.

,

. **(Professionally not recommended)**

3: (Terminal Emulator)

#!/bin/bash

#Script that displays today's date

echo "Today's date is:"

date +"%A, %B %-d, %Y"

:

Terminal emulator , **cd sdcard**

, **(directory/folder)**

. **mkdir script**

. **cd script**

.

root

. **quick office**

/ **Terminal emulator**

. ,

, , .
, 1
.

Today’s date is: Wednesday, October 1, 2014

4 : (Terminal Emulator)
,

.

.

.

```
<HTML>
<HEAD>
   <TITLE>
   The title of your page
   </TITLE>
</HEAD>

<BODY>
   Your page content goes here.
</BODY>
</HTML>
```

4 - 1: (Terminal Emulator)

```
#!/bin/bash
# make_page - A script to produce an HTML file
echo "<HTML>"
echo "<HEAD>"
echo "  <TITLE>"
```

```
echo "  The title of your page"
echo "  </TITLE>"
echo "</HEAD>"
echo ""
echo "<BODY>"
echo "  Your page content goes here."
echo "</BODY>"
echo "</HTML>"
```

, (

.)

```
main_page.sh > filename.html
```

HTML .

echo HTML

.

4 - 2 (Terminal Emulator)

echo

. cat

.

EOF End Of File .

```
#!/bin/bash
# make_page - A script to produce an HTML file
cat << _EOF_
<HTML>
<HEAD>
   <TITLE>
   The title of your page
   </TITLE>
</HEAD>

<BODY>
```

Your page content goes here.

</BODY>

</HTML>

EOF

, main_page.sh > filename.html

HTML .

: (Technical terms)

– Parent process

– child process

– orphan process

– daemon process

– running state

– stopping state

– zombie state

– sleeping state

– interruptable sleep state

– uninterruptable sleep state

– Folder or directory

(...)

– 4 ,



.

- 4

:

boot kernel

.

initrd.img .

/etc/inittab

.

/etc/fstab

.

: (Variable usage)

.

env environment variable

.

(user-defined variables). # env

.

:

:

(destruction scripts)

.

(root user)

.

5: (Terminal Emulator)

title

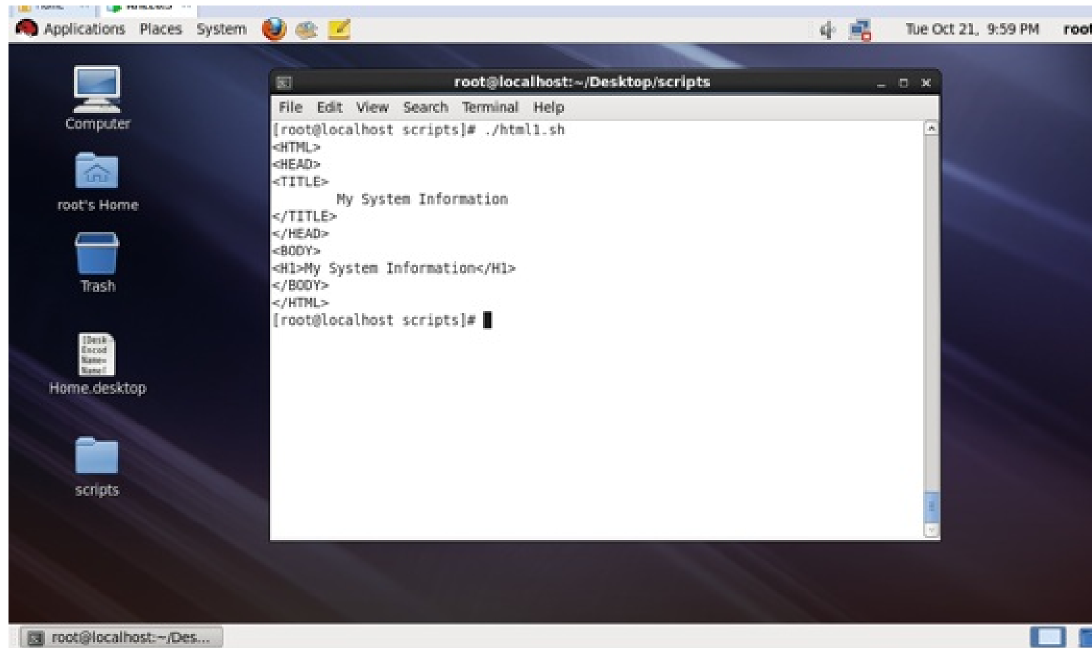
.

$title .

```
#!/bin/bash
# make_page - A script to produce an HTML file
title="My System Information"
cat<<- _EOF_
<HTML>
<HEAD>
<TITLE>
	$title
</TITLE>
</HEAD>
<BODY>
```

<H1>$title</H1>
</BODY>
</HTML>
EOF

. (chmod +x niral5.sh; ./niral5.sh)



:

.

6: (Root Terminal Emulator)

$HOSTNAME

environment variable .

#!/bin/bash

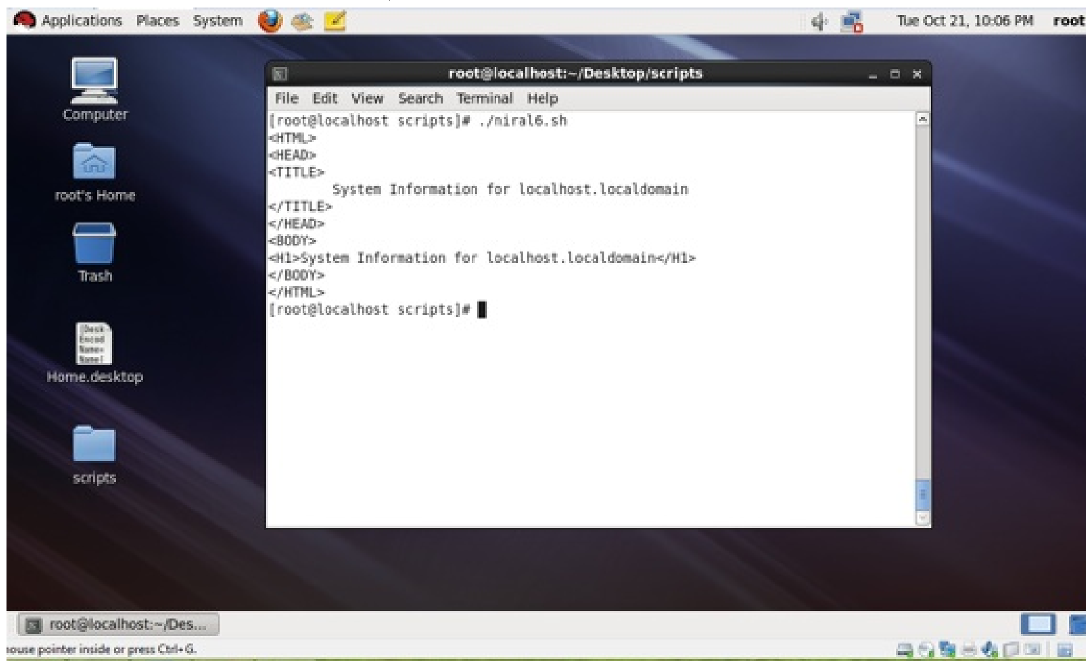
make_page - A script to produce an HTML file

title="System Information for"

cat<<- _EOF_

```
<HTML>
<HEAD>
<TITLE>
        $title $HOSTNAME
</TITLE>
</HEAD>
<BODY>
<H1>$title $HOSTNAME</H1>
</BODY>
</HTML>
_EOF_
```

. (chmod +x niral6.sh; ./niral6.sh)

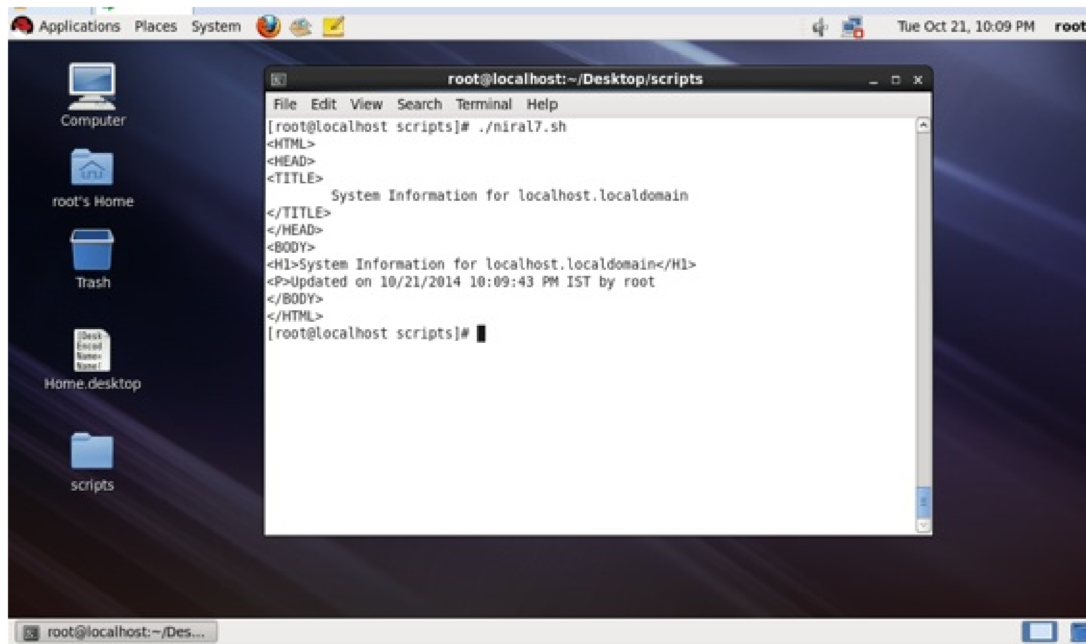


7: (Root Terminal Emulator)

date , HOSTNAME, USER

```
#!/bin/bash
# make_page - A script to produce an HTML file
TITLE="System Information for $HOSTNAME"
RIGHT_NOW=$(date +"%x %r %Z")
TIME_STAMP="Updated on $RIGHT_NOW by $USER"
cat<<- _EOF_
<HTML>
<HEAD>
<TITLE>
	$TITLE
</TITLE>
</HEAD>
<BODY>
<H1>$TITLE</H1>
<P>$TIME_STAMP
</BODY>
</HTML>
_EOF_
```

```
#chmod +x niral7.sh
#./niral7.sh
```



.

: (Technical terms)

– Operating system

– Booting stage

– First file

– kernel (core of the operating system)

– initrd file

– seven run levels of the operating system

- /etc/fstab file (file system table configuration file)

, - root user

- undo

- destruction script

- variables

- environment variables

- Substitution

(…)

– 5

மறுதொடக் கம்இயங் குநிலை ஏழே.
- நிரற்பா 5

:

.
init 0
. shutdown
. single user
multiuser
without network ,
multiuser with network , **future**
enhancement or unused ,
GUI-Graphical User Interface ,
restart .

init 0 – shutdown
init 1 – single user mode
init 2 – multi user mode without network
init 3 – multi user mode with network

init 4 – unused for future enhancement purpose

init 5 – XII or GUI mode

init 6 – reboot

.

#vim /etc/inittab

. Red Hat Enterprise Linux

3 5 . Ubunt

3 .

```
#
# inittab       This file describes how the INIT process shou
#               the system in a certain run-level.
#
# Author:       Miquel van Smoorenburg, <miquels@drinkel.nl.m
#               Modified for RHS Linux by Marc Ewing and Donn
#

# Default runlevel. The runlevels used by RHS are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not 
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:3:initdefault:


# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit
```

8

#!/bin/bash

make_page - A script to produce an HTML file

TITLE="System Information for $HOSTNAME"

RIGHT_NOW=$(date +"%x %r %Z")

```
TIME_STAMP="Updated on $RIGHT_NOW by $USER"
cat <<- _EOF_
   <HTML>
   <HEAD>
      <TITLE>
      $TITLE
      </TITLE>
   </HEAD>
   <BODY>
   <H1>$TITLE</H1>
   <P>$TIME_STAMP
   </BODY>
</HTML>
_EOF_
```

. RIGHT_NOW, TIME_STAMP
.

(updated)
.

9:

```
#!/bin/bash
# system_page - A script to produce an system information HTML file
##### Constants
TITLE="System Information for $HOSTNAME"
RIGHT_NOW=$(date +"%x %r %Z")
TIME_STAMP="Updated on $RIGHT_NOW by $USER"
##### Functions
function system_info
{
```

```
    # Temporary function stub
    echo "function system_info"
}
function show_uptime
{
    # Temporary function stub
    echo "function show_uptime"
}
function drive_space
{
    # Temporary function stub
    echo "function drive_space"
}
function home_space
{
    # Temporary function stub
    echo "function home_space"
}
##### Main
cat <<- _EOF_
  <html>
  <head>
      <title>$TITLE</title>
  </head>
  <body>
      <h1>$TITLE</h1>
      <p>$TIME_STAMP</p>
      $(system_info)
      $(show_uptime)
      $(drive_space)
```

```
      $(home_space)
  </body>
  </html>
_EOF_
function show_uptime
{
    echo "<h2>System uptime</h2>"
    echo "<pre>"
    uptime
    echo "</pre>"
}
function drive_space
{
    echo "<h2>Filesystem space</h2>"
    echo "<pre>"
    df
    echo "</pre>"
}
function home_space
{
    echo "<h2>Home directory space by user</h2>"
    echo "<pre>"
    echo "Bytes Directory"
    du -s /home/* | sort -nr
    echo "</pre>"
}
function system_info
{
    echo "<h2>System release info</h2>"
    echo "<p>Function not yet implemented</p>"
```

}

(Functions):

.

,

. (a function is a piece of a program, which is used to perform a particular task in the main program)

$(system_info)

$(show_uptime)

$(drive_space)

$(home_space)

. system_info

.

show_uptime ,

. drive_space

. home_space

user's home directory .

(usage of functions in shell script):

1. ஒரு குறிப்பிட்ட வேலையினை மட்டுமே செய்யப்பயன்படுகிறது.
2. மீண்டும் மீண்டும் எழுத வேண்டிய நிரல்வரிகளை ஒரே முறை எழுதி மீண்டும் மீண்டும் முதன்மை நிரலில் இருந்து அழைக்கலாம்.

3. நிரலர்களின் பணி எளிதாகிறது.
4. பொறியும் எளிதில் நிரல்வரிகளைப் புரிந்து குழப்பமின்றி செயல்பட உதவுகிறது.
5. வருங்கால நிரல் மேம்பாட்டிற்கு உதவுகிறது.
6. நிரலினை புதியவர்கள் படிக்கும் பொழுது, எளிதில் புரியும் வண்ணம் அமைகிறது.

:

– init 0 (shutdown or halt)

– single user (run-level 1)

– multiuser without network (run-level 2)

– multiuser with network (run-level 3)

– future enhancement (run-level 4)

– Graphical user interface (GUI run-level 5)

– restart (run-level 6)

- functions

( )

- 6

(Functions)

,

.

- 6

□□□□□□□□ □□□□□□□□□:

. .

10:

```
#!/bin/sh
# A simple script with a function...
add_a_user()
{
  USER=$1
  PASSWORD=$2
  shift; shift;
  # Having shifted twice, the rest is now comments ...
  COMMENTS=$@
```

```
  echo "Adding user $USER ..."
  echo useradd -c "$COMMENTS" $USER
  echo passwd $USER $PASSWORD
  echo "Added user $USER ($COMMENTS) with pass $PASSWORD"
}
###
# Main body of script starts here
###
echo "Start of script..."
add_a_user bob letmein Bob Holness the presenter
add_a_user fred badpassword Fred Durst the singer
add_a_user bilko worsepassword Sgt. Bilko the role model
echo "End of script..."
```

:

.

.

.

.

```
$1=bob
$2=letmein
$3=Bob
$4=Holness
$5=the
$6=presenter
```

11:

```
#!/bin/sh
myfunc()
{
  echo "I was called as : $@"
  x=2
}
### Main script starts here
echo "Script was called with $@"
x=1
echo "x is $x"
myfunc 1 2 3
echo "x is $x"
```

:

myfunc() .

(main program)

.

,

.

12:

```
#!/bin/sh
myfunc()
{
  echo "\$1 is $1"
  echo "\$2 is $2"
  # cannot change $1 - we'd have to say:
  # 1="Goodbye Cruel"
  # which is not a valid syntax. However, we can
```

```
  # change $a:
  a="Goodbye Cruel"
}

### Main script starts here
a=Hello
b=World
myfunc $a $b
echo "a is $a"
echo "b is $b"
```

:

11

. ,

.

:

.

filename.sh

./filename.sh

. Terminal emulator

,

.

.

sh –vx

./filename.sh

.

:

– functions

– functions

– complex instructions

– big task or complex task

– main program

– text editor

(...)

7 -

(if condition)

;

;

; .

- 7

:

if condition

. If condition ,

.

nested if statement .

, ++ nested if

statement . elif

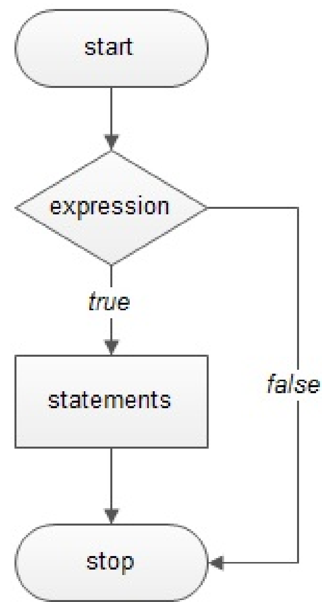
statement . elif statement

elif ladder

. , ++

.

.



,

.

.

.

if condition syntax:

First form

```
if condition ; then
    commands
fi
```

if condition syntax:

Second form

if *condition* ; then

commands

else

commands

fi

if condition syntax:

Third form

if *condition* ; then

commands

elif *condition* ; then

commands

fi

முதல்வகையான இருந்தால் கட்டளையினை, if statement இல்லாமலும் கையாளலாம். test என்ற keyword இங்கு if க்கு மாற்றாகப் பயன்படுத்தப்பட்டிருக்கிறது. இரண்டாவது # Second form இல் []சதுர அடைப்புக்குறிகள் பயன்படுத்தப்பட்டிருக்கின்றன.

First form

test *expression*

Second form

[*expression*]

:

13 **1:**

```
if [ -f .bash_profile ]; then
   echo "You have a .bash_profile. Things are fine."
else
   echo "Yikes! You have no .bash_profile!"
fi
```

if **condition**

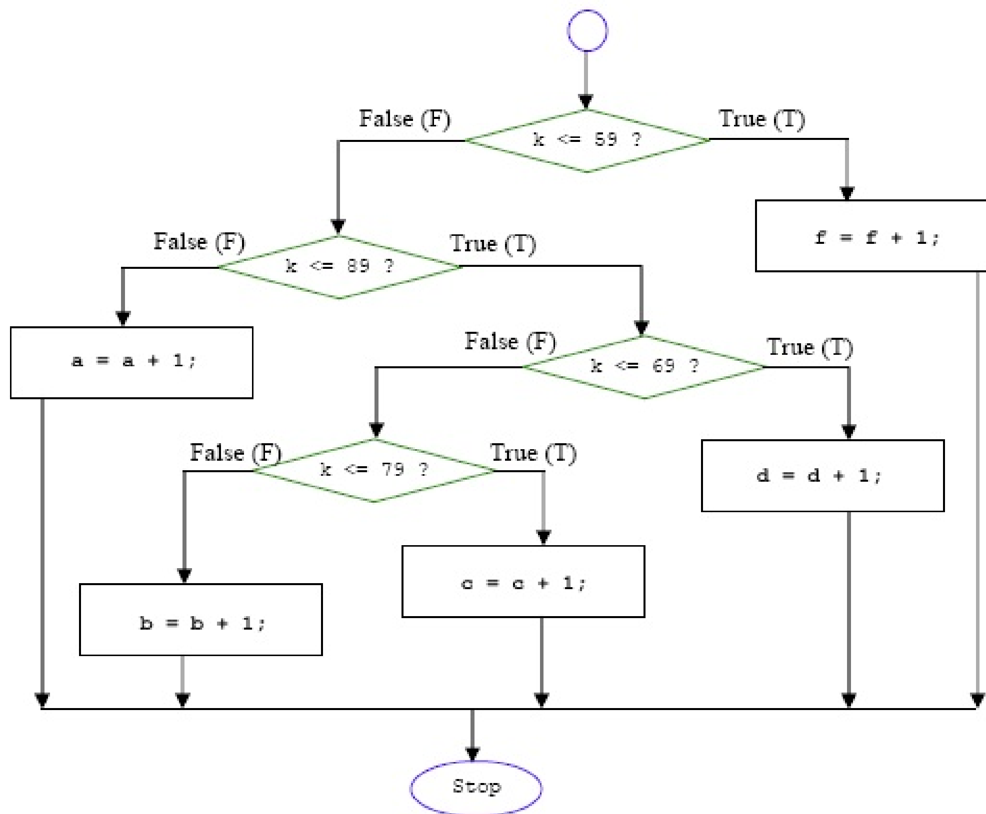
. **–f**

. **fi** **if** **condition**

. **{ } braces**

. **then**

keyword **.**



if statement

. decision making

.

elif ladder .

, if statement

,

.

Expression	Description
-d *file*	**True if *file* is a directory.**
-e *file*	**True if *file* exists.**
-f *file*	**True if *file* exists and is a regular file.**
-L *file*	**True if *file* is a symbolic link.**
-r *file*	**True if *file* is a file readable by you.**
-w *file*	**True if *file* is a file writable by you.**
-x *file*	**True if *file* is a file executable by you.**
file1* -nt *file2	**True if *file1* is newer than (according to modification time) *file2***
file1* -ot *file2	**True if *file1* is older than *file2***
-z *string*	**True if *string* is empty.**
-n *string*	**True if *string* is not empty.**
string1* = *string2	**True if *string1* equals *string2.***
string1* != *string2	**True if *string1* does not equal *string2.***

13 2:

```
if [ -f .bash_profile ]
then
   echo "You have a .bash_profile. Things are fine."
else
   echo "Yikes! You have no .bash_profile!"
fi
```

13 3:

```
if [ -f .bash_profile ]
then echo "You have a .bash_profile. Things are fine."
else echo "Yikes! You have no .bash_profile!"
fi
```

.

14

```
if [ $(id -u) = "0" ]; then
   echo "superuser"
fi
```

15

```
if [ $(id -u) != "0" ]; then
   echo "You must be the superuser to run this script" >&2
   exit 1
fi
```

பின்வரும் நிரலில், ஒரு நிரல் துண்டில் எவ்வாறு நாம் இருந்தால் கட்டளை வரியைப் பயன்படுத்துவது என்பது விளக்கப்பட்டுள்ளது.

16

```
function home_space
```

```
{
    # Only the superuser can get this information

    if [ "$(id -u)" = "0" ]; then
        echo "<h2>Home directory space by user</h2>"
        echo "<pre>"
        echo "Bytes Directory"
            du -s /home/* | sort -nr
        echo "</pre>"
    fi

}   # end of home_space
```

17

```
#!/bin/bash

number=1

if [ $number = "1" ]; then
    echo "Number equals 1
fi
```

18

```
#!/bin/bash

number=1

if [ $number = "1" ]; then
    echo "Number equals 1"
else
```

```
    echo "Number does not equal 1"
fi
```

19

```
#!/bin/bash

number=1

set -x
if [ $number = "1" ]; then
    echo "Number equals 1"
else
    echo "Number does not equal 1"
fi
set +x
```

.

:

If condition statement -

Elif statement -

Nested if statement -

Elif ladder -

(...)

– 8

(for loop)

.

- 8

:

(program statements),

( )

,

(get satisfied the given condition)

(loop)

(single or multiple statements)

(for statement) .

(If statement examples)

20:

```
#!/bin/bash

echo -n "Hurry up and type something! > "
if read -t 3 response; then
echo "Great, you made it in time!"
```

```
else
echo "Sorry, you are too slow!"
fi
```

read **-t** **3**

. **(system)**

(user input) .

if condition satisfied **Great, you made it in time!** . ,

Sorry, you are too slow! .

.

, **read** **-s** .

,

.

21:

```
#!/bin/bash

number=0
echo -n "Enter a number > "
read number

echo "Number is $number"
if [ $((number % 2)) -eq 0 ]; then
echo "Number is even"
else
echo "Number is odd"
fi
```

,

.

.

.

22:

```
#!/bin/bash

echo -n "Enter a number between 1 and 3 inclusive > "
read character
if [ "$character" = "1" ]; then
   echo "You entered one."
else
   if [ "$character" = "2" ]; then
      echo "You entered two."
   else
      if [ "$character" = "3" ]; then
         echo "You entered three."
      else
         echo "You did not enter a number"
         echo "between 1 and 3."
      fi
   fi
fi
```

இந்நிரல் ஒன்றிலிருந்து மூன்றிற்குள் (மூன்று மற்றும் ஒன்று உட்பட) ஏதேனும் ஒரு எண்ணை உள்ளீடாகக் கொடுத்து அது என்ன என்பதை விளக்குவதாக அமைக்கப்பட்டுள்ளது. இது இருந்தால் அடுக்குக்கட்டளையை (nested if statement) விளக்குவதற்காக கொடுக்கப்பட்டுள்ளது. இதே போன்று ஒரு நிரலமைவைப் பயன்படுத்தி, கொடுக்கப்பட்ட மூன்று எண்களில் பெரியது, சிறியது என்ன (find the biggest or smallest of given three numbers or find the second smallest and second largest numbers) என்பன போன்ற நிரல்களைச் செய்து

பார்க்கவும்.

23:

```
#!/bin/sh
# This is some secure program that uses security.
VALID_PASSWORD="secret" #this is our password.
echo "Please enter the password:"
read PASSWORD
if [ "$PASSWORD" == "$VALID_PASSWORD" ]; then
echo "You have access!"
else
echo "ACCESS DENIED!"
fi
```

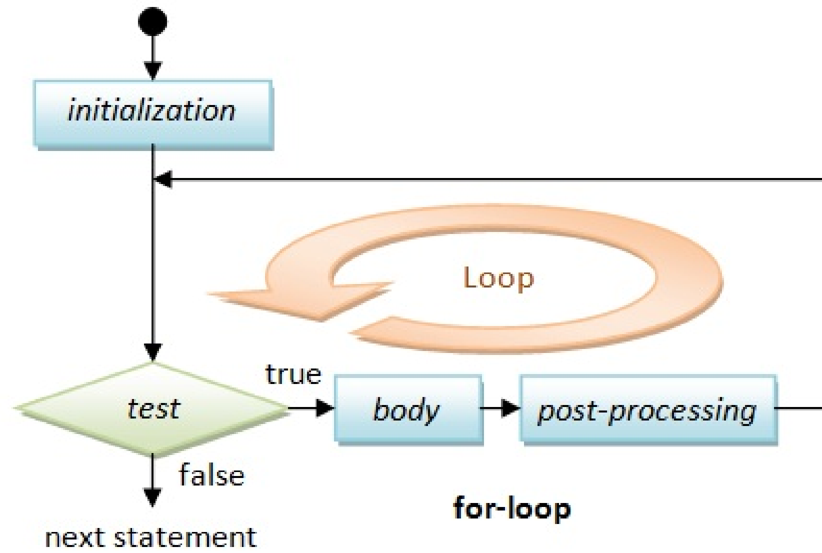
(password)

.

(string comparison script) .

.

.



(initialization) .

(testing the condition) ,

(body of the loop) .

,

.

.

(General syntax)

.

for var in word1 word2 ... wordN

do

Statement(s) to be executed for every word.

done

பின்வரும் இரண்டு நிரல்களைச் செய்து குறிப்பிட்ட வெளியீடுகள் கிடைக்கின்றனவா எனச் சரிபார்க்கவும்.

24:

```
#!/bin/bash
for var in 0 1 2 3 4 5 6 7 8 9
do
  echo $var
done
```

:

```
0
1
2
3
4
5
6
7
8
9
```

25:

```
#!/bin/sh
for FILE in $HOME/.bash*
do
  echo $FILE
done
```

:

```
/root/.bash_history
/root/.bash_logout
/root/.bash_profile
/root/.bashrc
```

(more for loop examples:)

26:

```
#!/bin/bash
for X in red green blue
do
echo $X
done
```

X

.

.

27:

```
#!/bin/bash
colour1="red"
colour2="light blue"
colour3="dark green"
for X in "$colour1" $colour2" $colour3"
do
echo $X
done
```

X

,

.

28:

```
#!/bin/bash
for X in *.html
do
grep -L '<UL>' "$X"
done
```

<UL>

,

. ,

,

UL

,

.

.

:

- Proper statements

- slow down

- running in a loop

- while condition is right, do the statements again and again

- for loop statement

– body of the loop

– variable

– directory or folder

– string or text

- loop

(…)

– 9

(for loop)

.

- 9

:

,

.

,

.

.

.

```
for ((i=0;i>=10;i++))
{
 Set of statements
}
```

.

.

29:

```
#!/bin/bash
# Random number generation using C language syntax
for (( i=1; i <= 5; i++ ))
do
 echo "Random number $i: $RANDOM"
done
```

இந்நிரலில் சிமொழியிலுள்ள அதே வகையான பொது அமைவு உள்ளது. { } என்னும் குறியீடுகளுக்கு (curly braces) மாற்றாக do, done என்னும் சொற்கள் உள்ளன. 5 முறை இந்த சுழற்சியானது இயக்கப்படுகிறது. வெளியீடு பின்வருமாறு அமைகிறது.

:

```
$ ./niral29.sh
Random number 1: 23320
Random number 2: 5070
Random number 3: 15202
Random number 4: 23861
Random number 5: 23435
```

30:

```
#!/bin/bash
# This program explains about infinite loop in shell script
i=1;
for (( ; ; ))
do
   sleep $i
   echo "Number: $((i++))"
```

```
done
```

இது ஒரு முடிவுறா சுழற்சி வளைவுக்கட்டளையாக அமைகிறது. ஆகக் கட்டளைக்குள் இருக்கும் கட்டளை வரிகள் திரும்பத்திரும்ப இயக்கப்பட்டு வெளியீடானது கிடைக்கிறது. இது சிமொழியில் உள்ள அதே வகையான அமைவாகும். இங்கு மாறியானது ஒன்று கூட்டப்பட்டு வருவதால், வளைவுக்கட்டளை எத்தனை முறை இயக்கப்பட்டது என்பதைக் காண இயலுமாறு வெளியீடு அமைகிறது.

:

```
$ ./niral30.sh
Number: 1
Number: 2
Number: 3
```

31:

```
#!/bin/bash
#for loop using comma

for ((i=1, j=10; i <= 5 ; i++, j=j+5))
do
 echo "Number $i: $j"
done
```

இந்நிரலிலும் சிமொழியில் உள்ள அதே வகையான அமைவு எடுத்தாளப்பட்டுள்ளது. இங்கு i மற்றும் j என்று இரண்டு வகையான மாறிகள் கையாளப்படுகின்றன. இரண்டும் தனித்தனியாக ஒன்றுடன் கூட்டப்படுகின்றன அதுவும் ஒரே ஒரு கட்டளை வரியில். வெளியீடானது பின்வருமாறு அமைகிறது. இங்கு இரண்டு மாறிகள் எடுத்துக்காட்டிற்காக கொடுக்கப்பட்டுள்ளன. இதே போல் பல மாறிகளை ஒரே கட்டளைவரியில் பயன்படுத்தி கணக்கீடுகளைச் செய்து வெளியீடுகளை எளிமையாகக் காணலாம்.

:

```
$ ./niral31.sh
Number 1: 10
Number 2: 15
Number 3: 20
Number 4: 25
Number 5: 30
```

32:

```
#!/bin/bash
#! For loop with range in numbers

for num in {1..10}
do
 echo "Number: $num"
done
```

இங்கு சிமொழியின் பொது அமைவு கையாளப்படவில்லை. ஆகக் கட்டளை கொண்டு ஒன்று முதல் பத்து வரை அச்சிட கட்டளை கொடுக்கப்பட்டுள்ளது.

:

```
$ ./niral32.sh
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
Number: 6
Number: 7
```

```
Number: 8
Number: 9
Number: 10
```

.

33:

```
#!/bin/bash
#for loop with with C language syntax
for ((i=1;i<=10;i++))
do
echo “Number: $i”
done
```

34:

```
#!/bin/bash
# Range of numbers with increments after “in” keyword

for num in {1..10..2}
do
 echo "Number: $num"
done
```

:

```
$ ./niral34.sh
Number: 1
Number: 3
Number: 5
Number: 7
Number: 9
```

இந்நிரலில் குறிப்பிட்ட மாறியானது, இரண்டு இரண்டாக தாவி ஓடுகிறது. இதே

நிரலை சிமொழியின் அமைவுடன் பின்வருமாறு எழுதி இயக்கலாம்.

35:

```
#!/bin/bash
#Niral 34 with C Language syntax
for ((i=1;i<=10;i=i+2))
do
echo ”Number: $i”
done
```

36:

```
#!/bin/bash
# fileinfo.sh  Fileinfo: operating on a file list contained in a variable
FILES="/usr/sbin/accept
/usr/sbin/pwck
/usr/sbin/chroot
/usr/bin/fakefile
/sbin/badblocks
/sbin/ypbind"     # List of files you are curious about.
                  # Threw in a dummy file, /usr/bin/fakefile.

echo

for file in $FILES
do
  if [ ! -e "$file" ]       # Check if file exists.
  then
    echo "$file does not exist."; echo
    continue                # On to next.
   fi
```

```
  ls -l $file | awk '{ print $9 "         file size: " $5 }'  # Print 2 fields.
  whatis `basename $file`   # File info.
  # Note that the whatis database needs to have been set up for this to work.
  # To do this, as root run /usr/bin/makewhatis.
  echo
done

exit 0
```

(file path) ,

,

.

.

:

– general syntax

– for loop

– shell script

– infinite for loop

- variable

(...)

-10



.

- 10

:

.

(condition)

.

(while loop – entry controlled loop).

.

,

.

(while loop - exit controlled loop)

.

.

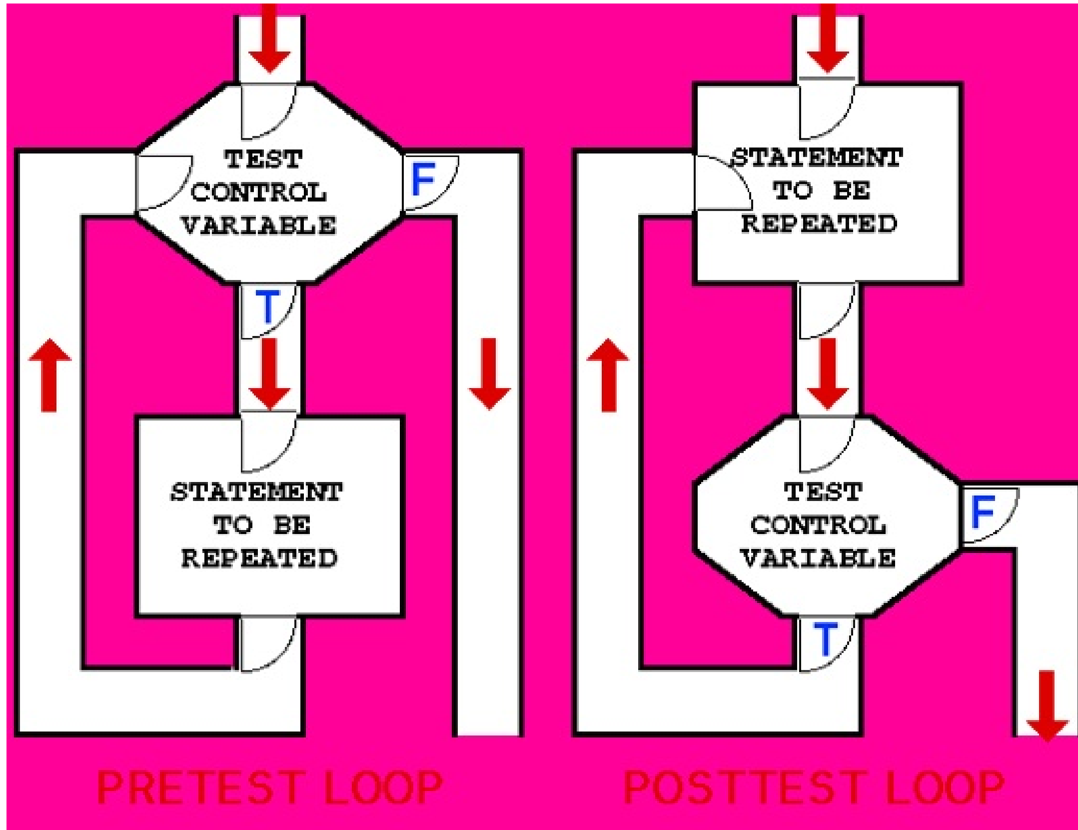
(pretest loop)

(post test loop) .

(whle loop vs until loop)

.

1. வரையிலும் கட்டளை சுழியல்லாத நிலை (non-zero status) வரை செய்யப்படுகிறது.
2. பொழுதெலாம் கட்டளை சுழி நிலை (zero status) வரை செய்யப்படுகிறது.
3. வரையிலும் கட்டளை எப்பொழுதுமே ஒரு முறை தனது கட்டளை வரிகளை இயக்கிவிடுகிறது.



(some more for statement examples):

37:

#!/bin/bash

```
# print the user names
i=1
for username in `awk -F: '{print $1}' /etc/passwd`
do
 echo "Username $((i++)) : $username"
done
```

:

(system)

(usernames)

. /etc/passwd

. awk

.

:

```
# ./niral36.sh
Username 1 : prasanna
Username 2 : arivu
Username 3 : nedilan
Username 4 : porrko
..
```

37:

```
#!/bin/bash
i=1
cd ~
for item in *
do
 echo "Item $((i++)) : $item"
done
```

:

. cd ~

.

:

```
# ./niral37.sh
Item 1 : positional-parameters.sh
Item 2 : backup.sh
Item 3 : emp-report.awk
Item 4 : item-list.sed
Item 5 : employee.db
Item 8 : storage
Item 9 : downloads
```

38:

```
#!/bin/bash
i=1
for file in /etc/[abcd]*.conf
do
 echo "File $((i++)) : $file"
done
```

:

(a,b,c,d

)

.

.

:

```
# ./niral38.sh
File 1 : /etc/asound.conf
File 2 : /etc/autofs_ldap_auth.conf
File 3 : /etc/cas.conf
File 4 : /etc/cgconfig.conf
File 5 : /etc/cgrules.conf
File 6 : /etc/dracut.conf
```

39:

```
#!/bin/bash
# Script for multiplication table for 5
#
if [ $# -eq 0 ]
then
echo "Error - Number missing form command line argument"
echo "Syntax : $0 number"
echo "Use to print multiplication table for given number"
exit 1
fi
n=5
for i in 1 2 3 4 5 6 7 8 9 10
do
echo "$n * $i = `expr $i \* $n`"
done
```

:

,

. n=5

,

.

:

```
# ./niral39.sh
```

5 * 1 = 5

5 * 2 = 10

...

..

5 * 10 = 50

(for loop another format)

.

(while loop)

.

:

– until loop

– while loop

– exit controlled

– entry controlled

– loop statements

- concept

– zero status

– non-zero status

– system

– username

– Home folder or home directory

(...)

-11

while loop – entry controlled loop

.

- 11

:

,

(whether the given condition is satisfied or not)

, (body of

the loop) . ,

.

, ,

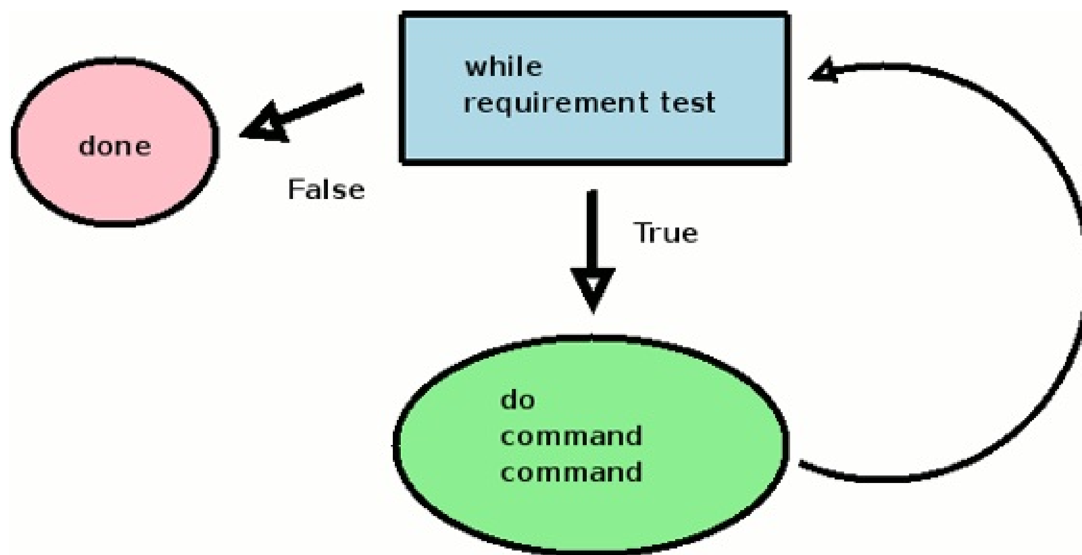
.

.

(zero)

, (condition is false)

.



.

40:

```
#!/bin/bash
#niral40
#chessboard
for (( i = 1; i <= 9; i++ )) ### Outer for loop ###
do
  for (( j = 1 ; j <= 9; j++ )) ### Inner for loop ###
  do
      tot=`expr $i + $j`
      tmp=`expr $tot % 2`
      if [ $tmp -eq 0 ]; then
         echo -e -n "\033[47m "
      else
         echo -e -n "\033[40m "
      fi
 done
 echo -e -n "\033[40m" #### set back background colour to black
```

```
 echo "" #### print the new line ###
done
```

:

(chess board) .

,

.

for ((i = 1; i <= 9; i++)) do	Begin the outer loop which runs 9 times., and the outer loop terminates when the value of i exceeds 9
for ((j = 1 ; j <= 9; j++)) do	Begins the inner loop, for each value of i the inner loop is cycled through 9 times, with the variable j taking values from 1 to 9. The inner for loop terminates when the value of j exceeds 9.
tot=`expr $i + $j` tmp=`expr $tot % 2`	See for even and odd number positions using these statements.
if [$tmp -eq 0]; then echo -e -n "\033[47m " else echo -e -n "\033[40m " fi	If even number position print the white colour block (using echo -e -n "\033[47m "statement); otherwise for odd position print the black colour box (using echo -e -n "\033[40m " statement). These statements are responsible to print entire chess board on screen with alternate colours.
done	End of inner loop

echo -e -n "\033[40m"	Make sure its black background as we always have on our terminals.
echo ""	Print the blank line
done	End of outer loop and shell scripts get terminated by printing the chess board.

:

#./niral40.sh



(Syntaxes of while loop):

while

.

.

Ksh shell syntax:

```
while [[ condition ]] ; do
command1
command1
commandN
done
```

csh syntax:

```
while command
do
  Statement(s) to be executed if command is true
done
```

Bash syntax:

```
while [ condition ]
do
command1
command2
commandN
done
```

(while)

.

41:

```
#!/bin/sh
#niral 41
a=0
while [ $a -lt 10 ]
do
  echo $a
  a=`expr $a + 1`
done
```

:

a ,

. lt

less than 10

(printing).

(an alternate for for command) .

:

#./niral41.sh
0
1
2
3
4
5
6
7
8
9

42:

```
#!/bin/bash
#niral 42
c=1
while [ $c -le 5 ]
do
echo "Welcome $c times"
(( c++ ))
done
```

:

(satisfying the given condition)

.

:

#./niral42.sh
Welcome 1 times

Welcome 2 times

Welcome 3 times

Welcome 4 times

Welcome 5 times

43:

```
#!/bin/bash
#niral 43
# This generates a file every 5 minutes
while true; do
touch pic-`date +%s`.jpg
sleep 300
done
```

:

while ,

.

true . touch

. sleep 300

(300)

.

:

#./niral43.sh

,

.

:

– entry controlled

– while loop

– body of the loop

– zero status, condition is false

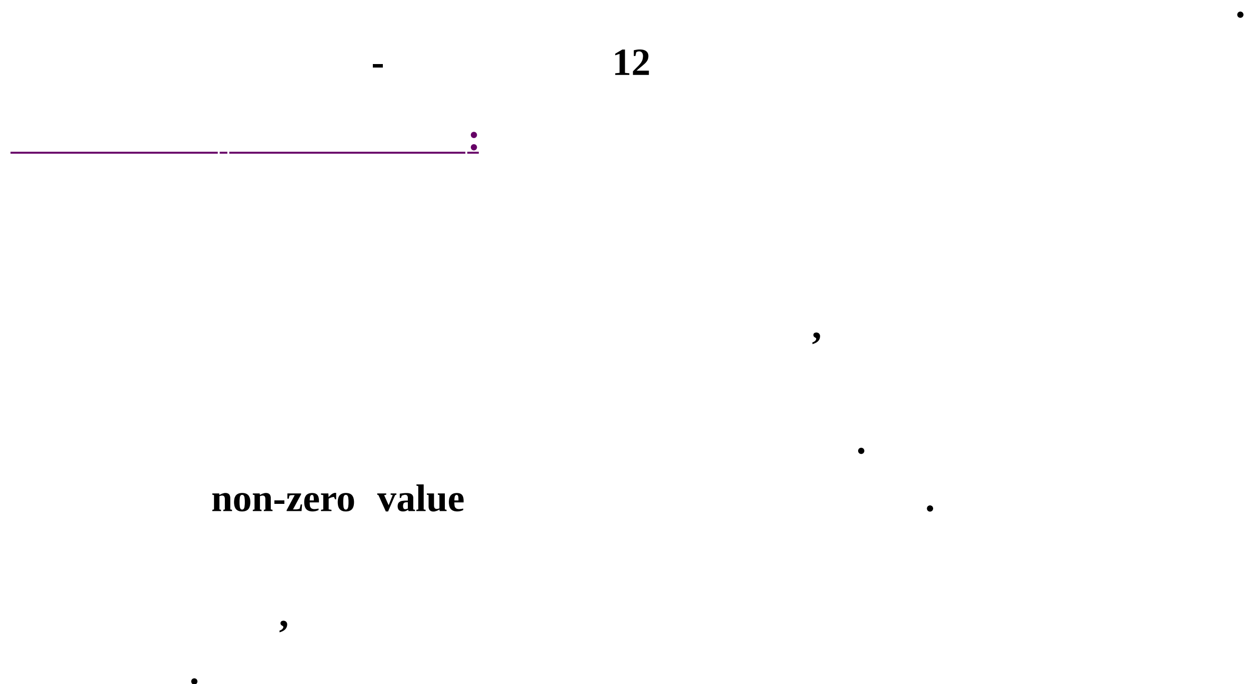
, , – zero

– chess board

(...)

-12

until loop – exit controlled loop



12 -

non-zero value

(General Syntax):

```
until expression
do
  commands #body of the loop
done
```

,

.

, (while loop command)

.

.

.

```
*until.sh (~) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
*until.sh
#until loop in action
count=1
until [ $count -ge 10 ]
do
        echo $count
        expr $count + 1
done
```

44:

```
# cat monitor.sh
#!/bin/bash
#niral 44
file=/tmp/logfile
until [ $(ls -l $file | awk '{print $5}') -gt 2000 ]
    do
      echo "Sleeping for next 5 seconds"
      sleep 5
    done
```

```
date=`date +%s`
cp $file "$file-"$date.bak
```

:

, log

. 2000

. .bak

.

.

/tmp .

:

```
# ./monitor.sh
Sleeping for next 5 seconds
Sleeping for next 5 seconds

# ls -l /tmp/logfile*
-rw-r--r-- 1 sss sss     2010 Jun 24 12:29 logfile
-rw-r--r-- 1 sss sss     2005 Jun 24 16:09 logfile-1277474574.bak
```

45:

```
# cat mac_wait.sh
#! /bin/bash
#niral  45
read -p "Enter IP Address:" ipadd
echo $ipadd
until ping -c 1 $ipadd
do
        sleep 60;
done
ssh $ipadd
```

:

, ssh ,

(remote access)

(ping command)

.

.

.

.

.

.

:

#./mac_wait.sh
Enter IP Address:192.143.2.10
PING 192.143.2.10 (192.143.2.10) 56(84) bytes of data.

--- 192.143.2.10 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

PING 192.143.2.10 (192.143.2.10) 56(84) bytes of data.
64 bytes from 192.143.2.10: icmp_seq=1 ttl=64 time=0.059 ms

--- 192.143.2.10 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.059/0.059/0.059/0.000 ms
The authenticity of host '192.143.2.10 (192.143.2.10)' can't be established.
Are you sure you want to continue connecting (yes/no)? yes

46:

```
#!/bin/bash
#niral 46
number=0
until [ $number -ge 10 ]; do
   echo "Number = $number"
   number=$((number + 1))
done
```

:

.

.

:

.

47:

```
#!/bin/bash
# This script copies files from my homedirectory into the webserver directory.
# A new directory is created every hour.
# If the pics are taking up too much space, the oldest are removed.

while true; do
DISKFUL=$(df -h $WEBDIR | grep -v File | awk '{print $5 }' | cut -d "%" -f1 -)

until [ $DISKFUL -ge "90" ]; do

       DATE=`date +%Y%m%d`
```

```
	HOUR=`date +%H`
	mkdir $WEBDIR/"$DATE"

	while [ $HOUR -ne "00" ]; do
		DESTDIR=$WEBDIR/"$DATE"/"$HOUR"
		mkdir "$DESTDIR"
		mv $PICDIR/*.jpg "$DESTDIR"/
		sleep 3600
		HOUR=`date +%H`
	done

DISKFULL=$(df -h $WEBDIR | grep -v File | awk '{ print $5 }' | cut -d "%" -f1 -)
done

TOREMOVE=$(find $WEBDIR -type d -a -mtime +30)
for i in $TOREMOVE; do
rm -rf "$i";
done

done
```

:

(home directory files)

.

(new directory)

.

,

.

:

.

:

– exit controlled

– until loop

– Boolean value (it may be true or false)

– non zero value

– at least once

– web server

– backup

– connection ping command

– ping replies

– self explanatory

(…)

-13.

(case statement)

.

-13

:

case

.

Case

, ,

.

. if else

(else if ladder) .

: case

esac .

(General syntax):

case word in

pattern1)

Statement(s) to be executed if pattern1 matches

```
    ;;
  pattern2)
    Statement(s) to be executed if pattern2 matches
    ;;
  pattern3)
    Statement(s) to be executed if pattern3 matches
    ;;
  *)
Default statement(s)
esac
```

* ,

(pattern)

.

48:

```
#!/bin/sh
#niral 48
FRUIT="kiwi"
case "$FRUIT" in
  "apple") echo "Apple pie is quite tasty."
  ;;
  "banana") echo "I like banana nut bread."
  ;;
  "kiwi") echo "New Zealand is famous for kiwi."
  ;;
esac
```

:

kiwi

,

.

(Output):

New Zealand is famous for kiwi.

49:

```
# cat filetype.sh
#!/bin/bash
#niral 49
for filename in $(ls)
do
# Take extension available in a filename
        ext=${filename##*\.}
        case "$ext" in
        c) echo "$filename : C source file"
          ;;
        o) echo "$filename : Object file"
          ;;
        sh) echo "$filename : Shell script"
           ;;
        txt) echo "$filename : Text file"
            ;;
        *) echo " $filename : Not processed"
          ;;
esac
done
```

:

(folder)

, (type of the file and extension)

. ,

, ,

, (file without extension or Not processed file) .

(Output):

```
# ./filetype.sh
a.c : C source file
b.c : C source file
c1.txt : Text file
fileop.sh : Shell script
obj.o : Object file
text : Not processed
t.o : Object file
```

50:

```
# cat yorno.sh
#niral50.sh
#!/bin/bash
echo -n "Do you agree with this? [yes or no]: "
read yno
case $yno in

        [yY] | [yY][Ee][Ss] )
                echo "Agreed"
                ;;

        [nN] | [n|N][O|o] )
                echo "Not agreed, you can't proceed the installation";
                exit 1
                ;;
        *) echo "Invalid input"
            ;;
esac
```

:

? ?

. (an

user is agreeing or disagreeing)

,

(checking the given text) .

(Output):

```
# ./yorno.sh
Do you agree with this? [yes or no]: YES
Agreed
```

51:

```
#!/bin/sh
# Wedding guest meals

# These variables hold the counters.
NUM_CHICKEN=0
NUM_STEAK=0
ERR_MSG=""

# This will clear the screen before displaying the menu.
clear

while :
do
# If error exists, display it
if [ "$ERR_MSG" != "" ]; then
echo "Error: $ERR_MSG"
echo ""
fi

# Write out the menu options...
echo "Chicken: $NUM_CHICKEN"
echo "Steak: $NUM_STEAK"
```

```
echo ""
echo "Select an option:"
echo " * 1: Chicken"
echo " * 2: Steak"
echo " * 3: Exit"

# Clear the error message
ERR_MSG=""

# Read the user input
read SEL

case $SEL in
1) NUM_CHICKEN=`expr $NUM_CHICKEN + 1` ;;
2) NUM_STEAK=`expr $NUM_STEAK + 1` ;;
3) echo "Bye!"; exit ;;
*) ERR_MSG="Please enter a valid option!"
esac

# This will clear the screen so we can redisplay the menu.
clear
done
```

:

,

(user menu)

. ,

,

.

.

:

– case statement

– default pattern

– pattern

– solution or result statements

– while statement

– if statement

– user menu

( )

-14.

(case statement vs else if ladder)



.

-14

:

.

.

.

.

.

,

.

.

(else if ladder command in shell script) .

52:

```
#!/bin/bash
#niral52.sh
#example for elif ladder in shell script
read -p "Enter value of i :" i

if [ $i -eq 5 ]
then
   echo "Value of i is 5"
elif [ $i -eq 10 ]
then
   echo "Value of i is 10"
elif [ $i -eq 20 ]
then
   echo "Value of i is 20"
elif [ $i -eq 30 ]
then
   echo "Value of i is 30"
else
   echo "Value of i is not equal to 5,10,20 or 30"
fi
```

:

.

.

,

. ,

.

(programmer)

,

. (optimal answers or optimal results) .

.

53:

```
#!/bin/bash
#niral53.sh
#script to find the greatest of the given three numbers
read -p "Enter value of i :" i
read -p "Enter value of j :" j
read -p "Enter value of k :" k
if [ $i -gt $j ]
then
   if [ $i -gt $k ]
   then
      echo "i is greatest"
   else
      echo "k is greatest"
   fi
else
   if [ $j -gt $k ]
   then
      echo "j is greatest"
   else
```

```
    echo "k is greatest"
    fi
fi
```

:

(nested if statement)

.

.

,

.

,

. ,

.

54:

```
#!/bin/bash
#niral54.sh
printf 'Which Linux distribution do you know? '
read DISTR
case $DISTR in
     ubuntu)
          echo "I know it! It is an operating system based on Debian."
          ;;
     centos|rhel)
          echo "Hey! It is my favorite Server OS!"
          ;;
     windows)
          echo "Very funny..."
          ;;
     *)
          echo "Hmm, seems i've never used it."
```

```
    ;;
esac
```

:

.

$DISTR

.

.

.

(if statement script)

. ,

;;

.

(experienced programmers)

(many if statements)

,

(only one case statement)

.

:

```
# ./testcase.sh
Which Linux distribution do you know? centos
Hey! It is my favorite Server OS!
# ./testcase.sh
Which Linux distribution do you know? rhel
Hey! It is my favorite Server OS!
# ./testcase.sh
```

```
Which Linux distribution do you know? ubuntu
I know it too! It is an operating system based on Debian.
# ./testcase.sh
Which Linux distribution do you know? windows
Very funny...
# ./testcase.sh
Which Linux distribution do you know? pfff
Hmm, seems i've never used it.
```

55: (multiple values)

```
#!/bin/bash
NOW=$(date +"%a")
case $NOW in
Mon)
echo "Full backup";;
Tue|Wed|Thu|Fri)
echo "Partial backup";;
Sat|Sun)
echo "No backup";;
*) ;;
esac
```

:

. |

. $NOW

.

(if statement) .

(small scripts)

.

(big scripts)

(case statement) (if statement)

.

:

– multiple values

– if statement

– case statement

– sure activity

- optimal answers

- nested if statement

- experienced programmers

( )

-15.

(break statement in loop)

.

-15

:

, ,

(loop statement – while, until or for loop)

,

(break command).

.

56:

```
#!/bin/sh
#niral56.sh
a=0
while [ $a -lt 10 ]
do
  echo $a
```

```
  if [ $a -eq 5 ]
  then
    break
  fi
  a=`expr $a + 1`
done
```

:

, 10

, 5

. 5

.

:

```
#./niral56.sh
0
1
2
3
4
5
```

57:

```
#!/bin/sh
#niral57.sh
for var1 in 1 2 3
do
  for var2 in 0 5
  do
    if [ $var1 -eq 2 -a $var2 -eq 0 ]
    then
      break 2
```

```
    else
      echo "$var1 $var2"
    fi
  done
done
```

:

(two different for statement).

(break command)

.

:

```
#./niral57.sh
1 0
1 5
```

58:

```
#!/bin/bash
#niral58.sh
# This script provides wisdom
# You can now exit in a decent way.
FORTUNE=/usr/games/fortune
while true; do
echo "On which topic do you want advice?"
echo "1.  politics"
echo "2.  startrek"
echo "3.  kernelnewbies"
echo "4.  sports"
echo "5.  bofh-excuses"
echo "6.  magic"
echo "7.  love"
```

```
echo "8.  literature"
echo "9.  drugs"
echo "10. education"
echo

echo -n "Enter your choice, or 0 for exit: "
read choice
echo
case $choice in
    1)
    $FORTUNE politics
    ;;
    2)
    $FORTUNE startrek
    ;;
    3)
    $FORTUNE kernelnewbies
    ;;
    4)
    echo "Sports are a waste of time, energy and money."
    echo "Go back to your keyboard."
    echo -e "\t\t\t\t -- \"Unhealthy is my middle name\" Soggie."
    ;;
    5)
    $FORTUNE bofh-excuses
    ;;
    6)
    $FORTUNE magic
    ;;
    7)
```

```
    $FORTUNE love
    ;;
    8)
    $FORTUNE literature
    ;;
    9)
    $FORTUNE drugs
    ;;
    10)
    $FORTUNE education
    ;;
    0)
    echo "OK, see you!"
    break
    ;;
    *)
    echo "That is not a valid choice, try a number from 0 to 10."
    ;;
esac
done
```

:

(beauty of the break statement),

,

.

echo

. (This echo will also

be executed upon input that causes break to be executed (when the user types "0")).

,

echo

(echo "That is not a valid choice, try a number from 0 to 10.") .

:

#./niral58.sh

.

:

– running sequentially

– loop statement

– jump and come out

– break

– break statement

– user input

– beauty

– control

– demonstration

- execution

( )

-16.

(continue statement in loop)

.

-16

:

,

, (continue statement)

,

.

(General syntax:)

.

continue

continue n

(examples):

```
...
..
for i in something
do
[ condition ] && continue
cmd1
cmd2

done
..
...
...
..
while true
do
[ condition1 ] && continue
cmd1
cmd2
[ condition2 ] && break
done
..
...
```

59:

```
#!/bin/bash
#niral61.sh
x=0
while [ $x -le 5 ]
do
   echo "Before continue : $x"
```

```
    x=`expr $x + 1`
    continue
    echo "After continue : $x"
done
echo "While loop finished"
```

:

. continue command echo "After continue : $x"

.

:

```
Before continue : 0
Before continue : 1
Before continue : 2
Before continue : 3
Before continue : 4
Before continue : 5
While loop finished
```

60:

```
#!/bin/sh
#niral 59.sh
#mysql backup script
#Must be run as the root user
MUSER="admin" # MySQL user
MHOST="192.168.1.100" # MySQL server ip
MPASS="MySQLServerPassword" # MySQL password

# format dd-mm-yyyy
NOW=$(date +"%d-%m-%Y")
```

```
# Backupfile path
BPATH=/backup/mysql/$NOW
# if backup path does not exists, create it
[ ! -d $BPATH ] && mkdir -p $BPATH
# get database name lists
DBS="$(/usr/bin/mysql -u $MUSER -h $MHOST -p$MPASS -Bse 'show
databases')"
for db in $DBS
do
# Bakcup file name
FILE="${BPATH}/${db}.gz"

# skip database backup if database name is adserverstats or mint
[ "$db" == "adserverstats"  ] && continue
[ "$db" == "mint"  ] && continue

# okay lets dump a database backup
   /usr/bin/mysqldump -u $MUSER -h $MHOST -p$MPASS $db | /bin/gzip
-9 > $FILE
done
```

:

.

.

adserverstats or mint

.

.

.

:

.

,

.

, .

61:

```
#!/bin/bash
#niral61.sh
# convert all domain names to a lowercase
DOMAINS="$(echo $@|tr '[A-Z]' '[a-z]')"
# Path to named.conf
NAMEDCONF="/var/named/chroot/etc/named.conf"
# Check named.conf for error
NAMEDCHEKCONF="/usr/sbin/named-checkconf -t /var/named/chroot/"
# Display usage and die
if [ $# -eq 0 ]
then
	echo "Usage: $0 domain1 domain2 ..."
	exit 1
fi

# okay use for loop to process all domain names passed
# as a command line args
for d in $DOMAINS
```

```
do
# if domain  already exits, skip the rest of the loop
	grep $d $NAMEDCONF >/dev/null
	if [ $? -eq 0 ]
	then
		echo "$d exits in in $NAMEDCONF, skiping ..."
		continue # skip it
	fi

	# else add domain to named.conf
	echo "Adding domain $d to $NAMEDCONF..."


	echo "zone \"${d}\" {" >> $NAMEDCONF
	echo "	type master;"  >> $NAMEDCONF
	echo "	file \"/etc/named/master.${d}\";"  >> $NAMEDCONF
	echo "	allow-transfer { slaveservers; };"  >> $NAMEDCONF
	echo "};"  >> $NAMEDCONF

	# Run named configuration file syntax checking tool
	$NAMEDCHEKCONF >/dev/null
	if [ $? -ne 0 ]   # error found?
	then
	 echo "**** Warning: named-checkconf - Cannot reload named due to
errors for $d ****"
	else
	 echo "**** Domain $d successfully added to $NAMEDCONF ****"

	fi
done
```

:

( ), ,
.
,
,

.
.
$#
.
.
#./niral61.sh SOMETHING.COM SOMETHING1.COM
:
. . .
.
.

□□□□□□□□□□□□□□□□□□□:

– given task
– carried out fully
– loop statement
– continue statement
– to be continued automatically
– backup
– command line arguments
( )

-17

(command line parameters or command line arguments)

.

-17

:

,

.

,

.

(General syntax):

,

.

1. $0 இது நிகழும் நிரலினைக் குறிக்கிறது. (The name the script was invoked with. This may be a basename without directory component,

or a path name. This variable is not changed with subsequent shift commands.)

2. $1, $2, $3, ... இது ஒன்று இரண்டு மூன்று என்று அளவுருக்களை அல்லது, தருமதிப்புக்களை எடுத்துக் கொள்கிறது. (The first, second, third, ... command line argument, respectively. The argument may contain whitespace if the argument was quoted, i.e. "two words".)
3. $# இது எத்தனை தருமதிப்புக்கள் உள்ளன என நிரலுக்குத் தருகிறது. இதில் $0 என்பது எடுத்துக்கொள்ளப்படுவதில்லை. (Number of command line arguments, not counting the invocation name $0)
4. $@ இது தருமதிப்புக்களை அதே வரிசையில் தருகிறது. ("$@" is replaced with all command line arguments, enclosed in quotes, i.e. "one", "two three", "four". Whitespace within an argument is preserved.)
5. $* இதுவும் தருமதிப்புக்களை தருகிறது. வெற்றிடத்தையும் (including spacebar) விட்டுவிடாமல் இது சேர்த்துக்கொள்கிறது. ($* is replaced with all command line arguments. Whitespace is not preserved, i.e. "one", "two three", "four" would be changed to "one", "two", "three", "four". This variable is not used very often, "$@" is the normal case, because it leaves the arguments unchanged.)

62:

```
#!/bin/bash
#niral62.sh
# use predefined variables to access passed arguments
#echo arguments to the shell
echo $1 $2 $3 ' -> echo $1 $2 $3'
```

We can also store arguments from bash command line in special array
args=("$@")
#echo arguments to the shell
echo ${args[0]} ${args[1]} ${args[2]} ' -> args=("$@"); echo ${args[0]} ${args[1]} ${args[2]}'

#use $@ to print out all arguments at once
echo $@ ' -> echo $@'

use $# variable to print out
number of arguments passed to the bash script
echo Number of arguments passed: $# ' -> echo Number of arguments passed: $#'

:

.

.

.

:

.

.

63:

#!/bin/bash

```
#niral63.sh
echo "Positional Parameters"
echo '$0 = ' $0
echo '$1 = ' $1
echo '$2 = ' $2
echo '$3 = ' $3
```

:

.

```
some_program word1 word2 word3
$0 would contain "some_program"
$1 would contain "word1"
$2 would contain "word2"
$3 would contain "word3"
```

:

.

```
#./niral63.sh one two three
```

64:

```
#!/bin/bash
#niral64.sh
if [ "$1" != "" ]; then
   echo "Positional parameter 1 contains something"
else
   echo "Positional parameter 1 is empty"
fi
```

:

.

:

```
#./niral64.sh
```

.

65:

```
#!/bin/bash
#niral65.sh
if [ $# -gt 0 ]; then
   echo "Your command line contains $# arguments"
else
   echo "Your command line contains no arguments"
fi
```

:

,

,

.

65:

.

```
#!/bin/bash
#niral66.sh
echo "You start with $# positional parameters"

# Loop until all parameters are used up
```

```
while [ "$1" != "" ]; do
    echo "Parameter 1 equals $1"
    echo "You now have $# positional parameters"

    # Shift all the parameters down by one
    shift
done
```

:

- command line parameters

- command line arguments

– parameters

– decorate or write

– customized

– personal script

– run time

– given value

- output

( )

-18.

(command line parameters or command line arguments) -

.

- 18

:

,

.

66:

. ,

.

HTML

.

,

(functions)

.

.

main

.

```
#!/bin/bash
#niral66.sh
# system_page - A script to produce a system information HTML file
##### Constants

TITLE="System Information for $HOSTNAME"
RIGHT_NOW=$(date +"%x %r %Z")
TIME_STAMP="Updated on $RIGHT_NOW by $USER"


##### Functions

function system_info
{
   echo "<h2>System release info</h2>"
   echo "<p>Function not yet implemented</p>"

}   # end of system_info
```

Command Line Arguments

- Shell scripting can accept command line arguments.
- Within a shell script, you can refer to these args as $1, $2, $3 and so
- $# - Number of command line arguments
- $* - Display all the arguments
- $0 – Name of the script


```
function show_uptime
{
   echo "<h2>System uptime</h2>"
   echo "<pre>"
   uptime
   echo "</pre>"

}   # end of show_uptime



function drive_space
{
   echo "<h2>Filesystem space</h2>"
   echo "<pre>"
```

```
    df
    echo "</pre>"

}   # end of drive_space


function home_space
{
    # Only the superuser can get this information

    if [ "$(id -u)" = "0" ]; then
        echo "<h2>Home directory space by user</h2>"
        echo "<pre>"
        echo "Bytes Directory"
        du -s /home/* | sort -nr
        echo "</pre>"
    fi

}   # end of home_space


function write_page
{
    cat <<- _EOF_
    <html>
        <head>
        <title>$TITLE</title>
        </head>
        <body>
        <h1>$TITLE</h1>
```

```
        <p>$TIME_STAMP</p>
        $(system_info)
        $(show_uptime)
        $(drive_space)
        $(home_space)
        </body>
    </html>
_EOF_

}



function usage
{
    echo "usage: system_page [[[-f file ] [-i]] | [-h]]"
}




##### Main

interactive=
filename=~/system_page.html

while [ "$1" != "" ]; do
    case $1 in
        -f | --file )           shift
                                filename=$1
                                ;;
        -i | --interactive )    interactive=1
```

```
                ;;
    -h | --help )     usage
                exit
                ;;
    * )              usage
                exit 1
  esac
  shift
done

# Test code to verify command line processing

if [ "$interactive" = "1" ]; then
echo "interactive is on"
else
echo "interactive is off"
fi
echo "output file = $filename"


# Write page (comment out until testing is complete)

# write_page > $filename
```

(mini-project) .

.

,

HTML

.

(major projects)

.

, (parameters)

(arguments)

.

:

– single user

– working for a long run

– usage, utilization

– user

– parameters

– mini project

– major projects

( )

-19.

(sleep command)

.

-நிரற்பா 19

:

,

(sleep command) .

General syntax: ( )

.

sleep NUMBER[SUFFIX]
Where SUFFIX may be:
s for seconds (the default)
m for minutes.
h for hours.
d for days.

sleep --help
sleep --version

,

(sleep state) .

To sleep for 5 seconds, use:

sleep 5

To sleep for 2 minutes, use:

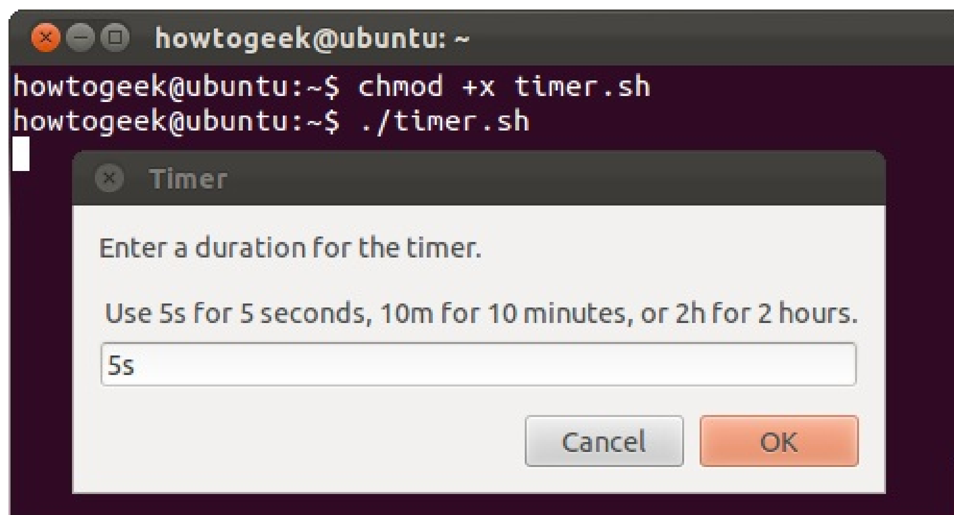
sleep 2m

To sleep for 3 hours, use:

sleep 3h

To sleep for 5 days, use:

sleep 5d



67:

```
#!/bin/bash
#niral67.sh
#this script explains how sleep works in shell script
echo "Hi, I'm sleeping for 5 seconds..."
sleep 5
echo "all Done."
```

:

.

, all Done .

./niral67.sh

68:

```
#!/bin/bash
#niral68.sh
## run commmand1, sleep for 1 minute and finally run command2 ##
command1 && sleep 1m && command2

## sleep in bash for loop ##
for i in {1..10}
do
  do_something_here
  sleep 5s
done
```

:

, sleep for

. do_something_here

.

69:

```
#!/bin/bash
#niral69.sh
# this script uses sleep command
## run while loop to display date and hostname on screen ##
while [ : ]
do
    clear
    tput cup 5 5
    date
```

```
   tput cup 6 5
   echo "Hostname : $(hostname)"
   sleep 1
done
```

:

./niral68.sh ,

(time)

(hostname)

(update)

. tput cup 5 5

. sleep 1

.

69:

```
#!/bin/bash
#niral69.sh
#run another script in the main script
i=1
while [ "$i" -ne 0 ]
do
  i=./runEmailAgent
  sleep 10
done
```

:

(script),

.

, ./runEmailAgent

.

.

70:

```
#!/bin/sh
#niral76.sh
#shell script example
before="$(date +%s)"
echo $before
sleep 3
after="$(date +%s)"
echo $after
elapsed_seconds="$(expr $after - $before)"
echo Elapsed time for code block: $elapsed_seconds
```

:

sleep

.

.

./niral70.sh

:

(Red Hat, Ubuntu, Suse Linux, Debian, Pinguy, Linux mint) .

:

– single user

– working for a long run

– usage, utilization

– user

– parameters

– mini project

– major projects

( )

-20.

(command line arguments or command line parameters)

!

- நிரற்பா 20

:

$

.

. (curly braces) { } .

$#

$@ ,

$$,

$? ,

( ) $!

.

71

```
#!/bin/bash
#niral71.sh
If [ $# = 3 ]
Then
echo “$@”
else
echo “Args are not three”
fi
```

:

,

.

.

72

```
#!/bin/bash
#niral72.sh
echo “Provide user name”
read i
a = `w | awk ‘{print $1}’ | sort –n | uniq | grep $i`
if [ “$i” = “$a” ]
then
echo “user logged in”
else
echo “user not logged in”
fi
```

:

,

,

.

a = `w | awk
‘{print $1}’ | sort –n | uniq | grep $i`

.

73 - 72 (2)

```
#!/bin/bash
#niral73.sh
echo –n “Please enter a username:”
read user
a=`grep $user /etc/passwd`
b=`w | awk ‘{print $1}’ | grep $user`
if [ ! “$a” ]
then
echo “$user is not a valid user”
exit
elif [ “$b” ]
then
echo “$user is logged in”
else
echo “$user is not logged in”
fi
```

:

72

. (arguments or parameters) (variable)

.

74

```
#!/bin/bash
#niral74.sh
#script used to file exists or not
a=$1
```

```
if [ -e “$a” ]
then
echo “$a file exist & modification time is” `ls –ltr $a | awk ‘{print $8}’`
else
echo “$a file doesn’t exist”
fi
```

:

,

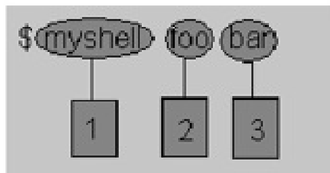
.

Why Command Line arguments required

1. Telling the command/utility which option to use.
2. Informing the utility/command which file or group of files to process (reading/writing of files).

Let's take rm command, which is used to remove file, but which file you want to remove and how you will tell this to rm command (even rm command don't ask you name of file that you would like to remove). So what we do is we write command as follows:
$ rm {file-name}
Here rm is command and filename is file which you would like to remove. This way you tail rm command which file you would like to remove. So we are doing one way communication with our command by specifying filename. Also you can pass command line arguments to your script to make it more users friendly. But how we access command line argument in our script.

Lets take ls command
$ Ls -a /*
This command has 2 command line argument -a and /* is another. For shell script,
$ myshell foo bar



1 Shell Script name i.e. myshell

2 First command line argument passed to myshell i.e. foo

3 Second command line argument passed to myshell i.e. bar

In shell if we wish to refer this command line argument we refer above as follows

1 myshell it is $0

2 foo it is $1

2 bar it is $2

75

#!/bin/bash

#niral75.sh

Call this script with at least 10 parameters, for example

./scriptname 1 2 3 4 5 6 7 8 9 10

MINPARAMS=10

```
echo

echo "The name of this script is \"$0\"."
# Adds ./ for current directory
echo "The name of this script is \"`basename $0`\"."
# Strips out path name info (see 'basename')

echo

if [ -n "$1" ]              # Tested variable is quoted.
then
 echo "Parameter #1 is $1"  # Need quotes to escape #
fi

if [ -n "$2" ]
then
 echo "Parameter #2 is $2"
fi

if [ -n "$3" ]
then
 echo "Parameter #3 is $3"
fi
# ...you can add your own comments here for further reference
if [ -n "${10}" ]  # Parameters > $9 must be enclosed in {brackets}.
then
 echo "Parameter #10 is ${10}"
fi

echo "-----------------------------------"
```

```
echo "All the command-line parameters are: "$*""


if [ $# -lt "$MINPARAMS" ]
then
  echo
    echo "This script needs at least $MINPARAMS command-line arguments!"
fi

echo
exit 0
```

:

. (# Call this script with at least 10 parameters, for example)

(output)

.

:

– Zero

– (first)

– last

– number in curly braces eg {333}

– call anything

( ) – count ($#)

($$) – current process id

- $@ arguments in the given order

– exit status

- **$! – Previous process id**

-

(**)**

-21

.

-நிரற்பா 21

:

,

,

.

76

```
#!/bin/bash
#niral76.sh
mkdir .recyclebin
mv $@ .recyclebin
echo –n "Please enter the filename to delete:"
read file
mv $file .recyclebin
```

:

(recycle bin) .

rm

. .recyclebin

.

recyclebin .

77

```
#!/bin/bash
#niral77.sh
a=$1
if [ “$a” = L ]
then
ls –l .recyclebin
fi
if [ “$a” = D ]
then
echo –n “Please enter the filename to delete:”
read file
mv $file .recyclebin
fi
```

:

. recycle bin

.

78

```
#!/bin/bash
#niral78.sh
#this program used to convert the image format into png

for i in *.pcx ; do
   CMD="convert -quality 625 $i `echo $i | sed -e 's/\.pcx$/.png/'`"
# Show the command-line to the user:
```

```
   echo $CMD
# Execute the command-line:
   eval $CMD
done
```

:

, ImageMagick

. ,

,

(changing to png format) .

.

79

```
#!/bin/bash
#niral79.sh
#To erase a file proper, requires writing random bytes into the disk blocks occupied by the file.
for i in * ; do
   dd if=/dev/urandom    \
     of="$i"             \
     bs=1024             \
     count=`expr 1 +    \
       \`stat "$i" | grep 'Size:' | awk '{print $2}'\`  \
         / 1024`
done
```

:

,

.

Random

. disk blocks

.

80

```
#!/bin/sh
#niral80.sh
# Get the files:
FILES=`ls -1`

for FILE in $FILES
do
IDX=`expr index $FILE .`

if [ "$IDX" == 0 ]; then
IDX=`expr length $FILE`
else
IDX=`expr $IDX - 1`
fi

SUB=`expr substr $FILE 1 $IDX`
echo "Sub File: $SUB"
done
```

:

, string

.

.

81

. ,

log

.

.

.

.

.

```
#!/bin/bash
# niral81.sh
#  Warning:
#  This script uses quite a number of features that will be explained
#+ later on.
#  By the time you've finished the first half of the book,
#+ there should be nothing mysterious about it.
LOG_DIR=/var/log
ROOT_UID=0     # Only users with $UID 0 have root privileges.
LINES=50       # Default number of lines saved.
E_XCD=86       # Can't change directory?
E_NOTROOT=87   # Non-root exit error.
# Run as root, of course.
if [ "$UID" -ne "$ROOT_UID" ]
then
  echo "Must be root to run this script."
  exit $E_NOTROOT
fi

if [ -n "$1" ]
# Test whether command-line argument is present (non-empty).
then
  lines=$1
```

```
else
  lines=$LINES # Default, if not specified on command-line.
fi
#  Stephane Chazelas suggests the following,
#+ as a better way of checking command-line arguments,
#+ but this is still a bit advanced for this stage of the tutorial.
#
#    E_WRONGARGS=85  # Non-numerical argument (bad argument
format).
#
#    case "$1" in
#    ""      ) lines=50;;
#    *[!0-9]*) echo "Usage: `basename $0` lines-to-cleanup";
#     exit $E_WRONGARGS;;
#    *       ) lines=$1;;
#    esac
#
#* Skip ahead to "Loops" chapter to decipher all this.
cd $LOG_DIR
if [ `pwd` != "$LOG_DIR" ]  # or   if [ "$PWD" != "$LOG_DIR" ]
                            # Not in /var/log?
then
  echo "Can't change to $LOG_DIR."
  exit $E_XCD
fi  # Doublecheck if in right directory before messing with log file.

# Far more efficient is:
#
# cd /var/log || {
#   echo "Cannot change to necessary directory." >&2
```

```
#   exit $E_XCD;
# }
tail -n $lines messages > mesg.temp # Save last section of message log file.
mv mesg.temp messages               # Rename it as system log file.
#  cat /dev/null > messages
#* No longer needed, as the above method is safer.
cat /dev/null > wtmp  #  ': > wtmp' and '> wtmp'  have the same effect.
echo "Log files cleaned up."
#  Note that there are other log files in /var/log not affected
#+ by this script.
exit 0
#  A zero return value from the script upon exit indicates success
#+ to the shell.
```

:

– proper time

– solving issues

– common tasks

– order of the commands

- package

( )

– 22

(shift command)

.

- நிரற்பா 22

:

,

,

. ,

.

Shell Scripting: Program Arguments : shift command

Shift

The shift command can be used to shift arguments to left side.

We can specify a count and we lose that many arguments on the left side. For a shift of 1, $2 becomes $1 and so on.

For a shift of 2, $3 will become $1.

It is useful to process arguments in a loop using a single variable to reference to argument one by one.

www.opengurukul.com 35

82

```
#!/bin/bash
#shift command in positional parameters
#niral82.sh
echo "Current command line args are: \$1=$1, \$2=$2, \$3=$3"
shift
echo "After shift command the args are: \$1=$1, \$2=$2, \$3=$3
```

:

.

.

,

,

.

Excute above script as follows:

chmod +x shiftdemo.sh

```
# ./shiftdemo -f foo bar
```

Current command line args are: $1=-f, $2=foo, $3=bar

After shift command the args are: $1=foo, $2=bar, $3=

83

```
#!/bin/bash
#niral83.sh
#using shift command
while [ "$1" ]
do
  if [ "$1" = "-b" ]; then
      ob="$2"
      case $ob in
        16) basesystem="Hex";;
         8) basesystem="Oct";;
         2) basesystem="bin";;
         *) basesystem="Unknown";;
      esac
     shift 2
  elif [ "$1" = "-n" ]
  then
    num="$2"
    shift 2
  else
    echo "Program $0 does not recognize option $1"
    exit 1
  fi
done
output=`echo "obase=$ob;ibase=10; $num;" | bc`
echo "$num Decimal number = $output in $basesystem number system(base=$ob)""
```

:

shift 2 ,

.

.

```
# chmod +x convert
# ./convert -b 16 -n 500
500 Decimal number = 1F4 in Hex number system(base=16)
# ./convert -b 8 -n 500
500 Decimal number = 764 in Oct number system(base=8)
# ./convert -b 2 -n 500
500 Decimal number = 111110100 in bin number system(base=2)
# ./convert -b 2 -v 500
Program ./convert does not recognize option -v
# ./convert -t 2 -v 500
Program ./convert does not recognize option -t
# ./convert -b 4 -n 500
500 Decimal number = 13310 in Unknown number system(base=4)
# ./convert -n 500 -b 16
500 Decimal number = 1F4 in Hex number system(base=16)
```

84:

```
#!/bin/bash
#niral84.sh
# This script can clean up files that were last accessed over 365 days ago.
```

```
USAGE="Usage: $0 dir1 dir2 dir3 ... dirN"

if [ "$#" == "0" ]; then
echo "$USAGE"
exit 1
fi

while (( "$#" )); do

if [[ $(ls "$1") == "" ]]; then
echo "Empty directory, nothing to be done."
  else
find "$1" -type f -a -atime +365 -exec rm -i {} \;
fi

shift

done
```

:

, 365

.

,

.

.

85:

```
#!/bin/bash
#niral85.sh
```

```
if [ $# -lt 1 ]; then
        echo "Usage: $0 package(s)"
        exit 1
fi
while (($#)); do
yum install "$1" << CONFIRM
y
CONFIRM
shift
done
```

:

. (This is used to install multiple packages at once.)

.

:

– in the given order

– available parameter

– usage of the script

– expanding the usage

– on emergency

– shift command

- folders

(   )

– 23 Trap command (-

)

.

- நிரற்பா 23

:

,

signal

(trap command)

.

Syntax ( )

.

trap arg signal

trap command signal

trap 'action' signal1 signal2 signalN

trap 'action' SIGINT

trap 'action' SIGTERM SIGINT SIGFPE SIGSTP

trap 'action' 15 2 8 20

86

```
#!/bin/bash
#niral86.sh
# capture an interrupt # 0
trap 'echo "Exit 0 signal detected..."' 0

# display something
echo "This is a test"

# exit shell script with 0 signal
exit 0
```

:

.

.

permission

.

```
chmod +x testtrap.sh
./testtrap.sh
```

:

```
This is a test
Exit 0 signal detected....
```

87

```
#!/bin/bash
#niral87.sh
# Capture an interrupt # 2 (SIGINT)
trap '' 2
```

read CTRL+C from keyboard with 30 second timeout

read -t 30 -p "I'm sleeping hit CTRL+C to exit..."

:

,

. CTRL+C ,

.

I'm sleeping hit CTRL+C to exit...^C^C^C^C

Command

- trap action after receiving signal

    `trap command signal`

- signal — explain

signal	explain
HUP (1)	hung up
INT (2)	interrupt (Crtl + C)
QUIT (3)	Quit (Crtl + \)
ABRT (6)	Abort
ALRM (14)	Alarm
TERM (15)	Terminate

42

88:

#!/bin/bash

#niral88.sh

Program to print a text file with headers and footers

TEMP_FILE=/tmp/printfile.txt

```
function clean_up {

# Perform program exit housekeeping
rm $TEMP_FILE
exit
}

trap clean_up SIGHUP SIGINT SIGTERM

lpr $1 > $TEMP_FILE
echo -n "Print file? [y/n]: "
read
if [ "$REPLY" = "y" ]; then
lpr $TEMP_FILE
fi
clean_up
```

:

, header
footer . clean_up
(function),
.

.

89:

```
#!/bin/bash
#niral89.sh
#another version of the previous script
# Program to print a text file with headers and footers
```

```
# Usage: print file

# Create a temporary file name that gives preference
# to the user's local tmp directory and has a name
# that is resistant to "temp race attacks"

if [ -d "~/tmp" ]; then
TEMP_DIR=~/tmp
else
TEMP_DIR=/tmp
fi
TEMP_FILE=$TEMP_DIR/printfile.$$.$RANDOM
PROGNAME=$(basename $0)

function usage {

# Display usage message on standard error
echo "Usage: $PROGNAME file" 1>&2
}

function clean_up {

# Perform program exit housekeeping
# Optionally accepts an exit status
rm -f $TEMP_FILE
exit $1
}

function error_exit {
```

```
# Display error message and exit
echo "${PROGNAME}: ${1:-"Unknown Error"}" 1>&2
clean_up 1
}

trap clean_up SIGHUP SIGINT SIGTERM

if [ $# != "1" ]; then
usage
error_exit "one file to print must be specified"
fi
if [ ! -f "$1" ]; then
error_exit "file $1 cannot be read"
fi

lpr $1 > $TEMP_FILE || error_exit "cannot format file"

echo -n "Print file? [y/n]: "
read
if [ "$REPLY" = "y" ]; then
lpr $TEMP_FILE || error_exit "cannot print file"
fi
clean_up
```

:

. function usage, function clean_up, function error_exit

.

, ,

.

,

.

:

– signal

– trap command

– particular action

– interrupt in between

– house keeping

(   )

– 24 (getopts command)

.

- நிரற்பா 24

:

,

.

while loop

. (This command is used to check valid command line argument are passed to script. Usually with while loop.)

.

,

,

.

Syntax ( )

.

getopts {optsring} {variable1}

90

```
#!/bin/bash
#go.sh
#niral90.sh
while getopts ":a" opt; do
  case $opt in
    a)
      echo "-a was triggered!" >&2
      ;;
    \?)
      echo "Invalid option: -$OPTARG" >&2
      ;;
  esac
done
```

:

.

. (Calling it without any arguments)

```
# ./go_test.sh
```

(Nothing happened? Right. getopts didn't see any valid or invalid options (letters preceded by a dash), so it wasn't triggered.)

. (Calling it with non-option arguments)

./go_test.sh /etc/passwd

. (Again — nothing happened.

The very same case: getopts didn't see any valid or invalid options (letters preceded by a dash), so it wasn't triggered.)

The arguments given to your script are of course accessible as $1 - ${N}.

Calling it with option-arguments

Now let's trigger getopts: Provide options.

.

./go_test.sh -b

Invalid option: -b

.

.

As expected, getopts didn't accept this option and acted like told above: It placed? into $opt and the invalid option character (b) into $OPTARG. With our case statement, we were able to detect this.

Now, a valid one (-a):

.

./go_test.sh -a

-a was triggered!

You see, the detection works perfectly. The a was put into the variable $opt for our case statement.

Of course it's possible to mix valid and invalid options when calling:

./go_test.sh -a -x -b -c

-a was triggered!

```
Invalid option: -x
Invalid option: -b
Invalid option: -c
```

,

.

.

```
# ./go_test.sh -a -a -a -a
-a was triggered!
-a was triggered!
-a was triggered!
-a was triggered!
```

91

```
 #!/bin/bash
#niral91.sh
# Usage: ani -n -a -s -w -d
# help_ani() To print help
help_ani()
{
  echo "Usage: $0 -n -a -s -w -d"
  echo "Options: These are optional argument"
  echo " -n name of animal"
  echo " -a age of animal"
  echo " -s sex of animal "
  echo " -w weight of animal"
  echo " -d demo values (if any of the above options are used "
  echo " their values are not taken)"
  exit 1
}
#
```

```
#Set default value for variable
#
isdef=0
na=Moti
age="2 Months" # may be 60 days, as U like it!
sex=Male
weight=3Kg
#
#if no argument
#
if [ $# -lt 1 ]; then
  help_ani
fi
while getopts n:a:s:w:d opt
do
  case "$opt" in
    n) na="$OPTARG";;
    a) age="$OPTARG";;
    s) sex="$OPTARG";;
    w) weight="$OPTARG";;
    d) isdef=1;;
    \?) help_ani;;
  esac
done
if [ $isdef -eq 0 ]
then
   echo "Animal Name: $na, Age: $age, Sex: $sex, Weight: $weight (user define mode)"
else
  na="Pluto Dog"
```

```
  age=3
  sex=Male
  weight=20kg
  echo "Animal Name: $na, Age: $age, Sex: $sex, Weight: $weight (demo mode)"
fi
```

:

, ,

, (demo)

.

We have script called ani which has syntax as

ani -n -a -s -w -d

Options: These are optional argument

-n name of animal

-a age of animal

-s sex of animal

-w weight of animal

-d demo values (if any of the above options are used their values are not taken)

:

Save it and run as follows

```
# chmod +x ani
# ani -n Lassie -a 4 -s Female -w 20Kg
# ani -a 4 -s Female -n Lassie -w 20Kg
# ani -n Lassie -s Female -w 20Kg -a 4
# ani -w 20Kg -s Female -n Lassie -a 4
# ani -w 20Kg -s Female
```

ani -n Lassie -a 4
ani -n Lassie
ani -a 2
:
,
.
. ( )
.
.
ani -nLassie -a4 -sFemal -w20Kg
space option
.
, .
ani -nLassie-a4-sFemal-w20Kg
ani -n Lassie -a 4 -s Female -w 20Kg -c Mammal
-c is not one of the valid options.

:

– parameter
– sequence system
– check and validate
– script
– output or answer
– getopts command
- beginners

(   )

– 25 (cut command)

.

- நிரற்பா 25

:

,

,

(columns)

.

Syntax (   )

.

cut –d(delimiter) f(field no) filename

eg: cut –d: f1 /etc/passwd

91

(data.txt)

.

one two three four five

alpha beta gamma delta epsilon

```
#!/bin/bash
#niral91.sh
#cut statement in the file filtering
cut -f 3 data.txt
```

:

.

```
three
gamma
```

92

```
#!/bin/bash
#niral91.sh
#cut statement
cut -f 1-2,4-5 data.txt
```

:

, ,

.

```
one two four five
alpha beta delta epsilon
```

93

```
#!/bin/bash
#niral93.sh
#cut statement example 3
cut -f 1 -d ':' /etc/passwd
```

:

delimiter ( )

. :

. f1

.

root

daemon

bin

sys

chope

```
TechFundaes
File  Edit  View  Terminal  Tabs  Help
[gatz@gbox]>cat matrixSpace.txt
name friends fav_apps timeline_enabled photos date_of_joining
neo 1252 give_hearts yes 151 05_JUN_03
morpheus 987 texas_poker no 245 12_APR_05
trinity 1562 angry_bird no 91 09_SEP_05
[gatz@gbox]>cut -d' ' -f-3 matrixSpace.txt
name friends fav_apps
neo 1252 give_hearts
morpheus 987 texas_poker
trinity 1562 angry_bird
[gatz@gbox]>
```

94

```
#!/bin/bash
#niral94.sh
#delimiter example no 2
cut -f 1,3 -d ':' --output-delimiter=$'\t' /etc/passwd
```

:

tab

.

root 0

daemon 1

bin 2

sys 3

chope 1000

95

```
#!/bin/bash
#niral95.sh
#combine cut command with other unix command
ps axu | grep python | sed 's/\s\+/ /g' | cut -d' ' -f2,11-
```

:

.

.

.

2231 /usr/bin/python /usr/lib/unity-lens-video/unity-lens-video

2311 /usr/bin/python /usr/lib/unity-scope-video-remote/unity-scope-video-remote

2414 /usr/bin/python /usr/lib/ubuntuone-client/ubuntuone-syncdaemon

2463 /usr/bin/python /usr/lib/system-service/system-service-d

3274 grep --color=auto python

:

– data

– columns or fields

– cut command

– system admins

( ) – **delimiter**

- separator

( )

– 26 (paste command)

.

- நிரற்பா 26

:

,

,

.

Syntax ( )

paste [OPTION]... [FILE]…

Options are as follows:

-d, --delimiters=LIST reuse characters from LIST instead of tabs.

-s, --serial paste one file at a time instead of in parallel.

--help Display a help message, and exit.

--version Display version information, and exit.

paste file1.txt file2.txt

.

96

#!/bin/bash

#niral96.sh

cat file1

Unix

Linux

Windows

cat file2

Dedicated server

Virtual server

cat file3

Hosting

Machine

Operating system



:

.

97

#!/bin/bash

```
#niral97.sh
paste file1 file2
Unix    Dedicated server
Linux   Virtual server
Windows

paste file2 file1
Dedicated server  Unix
Virtual server    Linux
                  Windows
```

:

.

98

```
#!/bin/bash
#niral98.sh
paste -d"|" file1 file2
Unix|Dedicated server
Linux|Virtual server
Windows|
```

:

|

. |

.

99

```
#!/bin/bash
#niral99.sh
paste -s file1 file2
Unix    Linux   Windows
Dedicated server        Virtual server
```

:

.

100

```
#!/bin/bash
#niral100.sh
paste -d"|," file1 file2 file3
Unix|Dedicated server,Hosting
Linux|Virtual server,Machine
Windows|,Operating system
cat file1 | paste - -
Unix    Linux
Windows
```

:

,

.

:

– paste command

– data

– text or string

– different files

( )

– 27 (tput command)

.

• நிரற்பா 27

:

terminal

(maximize or minimize)

(tput)

.

Examples ( )

tput longname

tput -T screen longname

tput colors

tput cols

tput bce && echo "True"

paste file1.txt file2.txt

```
#!/bin/bash
#niral101.sh
alias term_size=`echo "Rows=$(tput lines) Cols=$(tput cols)"'
   # term_size2 - Dynamically display terminal window size

   redraw() {
      clear
      echo "Width = $(tput cols) Height = $(tput lines)"
   }

   trap redraw WINCH

   redraw
   while true; do
      :
   done
```

:

SIGWINCH (signal)

.

:

```
Width = 80 Height = 24
```

(Controlling the Cursor)

Capname Description

sc Save the cursor position

rc Restore the cursor position

home Move the cursor to upper left corner (0,0)

cup <row> <col> Move the cursor to position row, col

cud1 Move the cursor down 1 line

cuu1 Move the cursor up 1 line

civis Set to cursor to be invisible

cnorm Set the cursor to its normal state

102

```
#!/bin/bash
#niral102.sh
  # term_size3 - Dynamically display terminal window size
  #            with text centering

  redraw() {
    local str width height length
```

```
    width=$(tput cols)
    height=$(tput lines)
    str="Width = $width Height = $height"
    length=${#str}
    clear
    tput cup $((height / 2)) $(((width / 2) - (length / 2)))
    echo "$str"
}

trap redraw WINCH

redraw
while true; do
    :
done
```

:

```
Width = 80 Height = 24
```

.

Capname Description

bold Start bold text

smul Start underlined text

rmul End underlined text

rev Start reverse video

blink Start blinking text

invis Start invisible text

smso Start "standout" mode

rmso End "standout" mode

sgr0 Turn off all attributes

setaf <value> Set foreground color

setab <value> Set background color

103

```
#!/bin/bash
#niral103.sh
    # tput_characters - Test various character attributes

    clear

    echo "tput character test"
    echo "==================="
    echo

    tput bold;  echo "This text has the bold attribute.";     tput sgr0

    tput smul;  echo "This text is underlined (smul).";       tput rmul

    # Most terminal emulators do not support blinking text (though xterm
```

```
# does) because blinking text is considered to be in bad taste ;-)
tput blink; echo "This text is blinking (blink).";      tput sgr0

tput rev;   echo "This text has the reverse attribute";   tput sgr0

# Standout mode is reverse on many terminals, bold on others.
tput smso;  echo "This text is in standout mode (smso)."; tput rmso

tput sgr0
echo
```

:

```
tput character test
===================

This text has the bold attribute.
This text is underlined (smul).
This text is blinking (blink).
This text has the reverse attribute
This text is in standout mode (smso).

me@linuxbox ~ $
```

.

.

.

_____________________:

– **tput command**

– **maximize or minimize**

– **text or string**

– **different files**

(   )

– 28 (tput command)

.

- நிரற்பா 28

:

.

.

(Text color)

Value Color

0 Black

1 Red

2 Green

3 Yellow

4 Blue

5 Magenta

6 Cyan

7 White

8 Not used

9 Reset to default color

104

```
#!/bin/bash
#script104.sh
  # tput_colors - Demonstrate color combinations.

  for fg_color in {0..7}; do
    set_foreground=$(tput setaf $fg_color)
    for bg_color in {0..7}; do
      set_background=$(tput setab $bg_color)
      echo -n $set_background$set_foreground
      printf ' F:%s B:%s ' $fg_color $bg_color
    done
    echo $(tput sgr0)
  done
```

:

```
me@linuxbox ~ $ tput_colors
          F:0 B:1  F:0 B:2  F:0 B:3  F:0 B:4  F:0 B:5  F:0 B:6  F:0 B:7
 F:1 B:0           F:1 B:2  F:1 B:3  F:1 B:4  F:1 B:5  F:1 B:6  F:1 B:7
 F:2 B:0  F:2 B:1           F:2 B:3  F:2 B:4  F:2 B:5  F:2 B:6  F:2 B:7
 F:3 B:0  F:3 B:1  F:3 B:2           F:3 B:4  F:3 B:5  F:3 B:6  F:3 B:7
 F:4 B:0  F:4 B:1  F:4 B:2  F:4 B:3           F:4 B:5  F:4 B:6  F:4 B:7
 F:5 B:0  F:5 B:1  F:5 B:2  F:5 B:3  F:5 B:4           F:5 B:6  F:5 B:7
 F:6 B:0  F:6 B:1  F:6 B:2  F:6 B:3  F:6 B:4  F:6 B:5           F:6 B:7
 F:7 B:0  F:7 B:1  F:7 B:2  F:7 B:3  F:7 B:4  F:7 B:5  F:7 B:6
me@linuxbox ~ $
```

:

.

105

(Clearing the Screen)

.

Capname Description

smcup Save screen contents (
.)

rmcup Restore screen contents (
.)

el Clear from the cursor to the end of the line (
.)

el1 Clear from the cursor to the beginning of the line (
.)

ed Clear from the cursor to the end of the screen (
)

clear Clear the entire screen and home the cursor (
.)

```
#!/bin/bash
#niral105.sh
    # tput_menu: a menu driven system information program

    BG_BLUE="$(tput setab 4)"
    BG_BLACK="$(tput setab 0)"
    FG_GREEN="$(tput setaf 2)"
    FG_WHITE="$(tput setaf 7)"

    # Save screen
```

```
tput smcup

# Display menu until selection == 0
while [[ $REPLY != 0 ]]; do
  echo -n ${BG_BLUE}${FG_WHITE}
  clear
  cat <<- _EOF_
    Please Select:

    1. Display Hostname and Uptime
    2. Display Disk Space
    3. Display Home Space Utilization
    0. Quit

_EOF_

  read -p "Enter selection [0-3] > " selection

  # Clear area beneath menu
  tput cup 10 0
  echo -n ${BG_BLACK}${FG_GREEN}
  tput ed
  tput cup 11 0

  # Act on selection
  case $selection in
    1)  echo "Hostname: $HOSTNAME"
        uptime
        ;;
    2)  df -h
```

```
      ;;
    3)  if [[ $(id -u) -eq 0 ]]; then
          echo "Home Space Utilization (All Users)"
          du -sh /home/* 2> /dev/null
        else
          echo "Home Space Utilization ($USER)"
          du -s $HOME/* 2> /dev/null | sort -nr
        fi
        ;;
    0)  break
        ;;
    *)  echo "Invalid entry."
        ;;
  esac
  printf "\n\nPress any key to continue."
  read -n 1
done

# Restore screen
tput rmcup
echo "Program terminated."
```

:

```
Please Select:

1. Display Hostname and Uptime
2. Display Disk Space
3. Display Home Space Utilization
0. Quit

Enter selection [0-3] > 2


Filesystem      Size  Used Avail Use% Mounted on
rootfs          5.8G  4.0G  1.6G  73% /
/dev/root       5.8G  4.0G  1.6G  73% /
devtmpfs        215M     0  215M   0% /dev
tmpfs            44M  260K   44M   1% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs            88M     0   88M   0% /run/shm
/dev/mmcblk0p5   60M  9.6M   50M  17% /boot


Press any key to continue.
```

.

,

.

:

– tput command

– terminal

– colors

- colors

- clearing

(   )

– 29 (tput command)

- நிரற்பா 29

:

tput

,

(terminal clock) (tensor)

(output) .

106

```
#niral106
#!/bin/bash

  # tclock - Display a clock in a terminal

  BG_BLUE="$(tput setab 4)"
  FG_BLACK="$(tput setaf 0)"
  FG_WHITE="$(tput setaf 7)"

  terminal_size() { # Calculate the size of the terminal
```

```
  terminal_cols="$(tput cols)"
  terminal_rows="$(tput lines)"
}

banner_size() {

  # Because there are different versions of banner, we need to
  # calculate the size of our banner's output

  banner_cols=0
  banner_rows=0

  while read; do
    [[ ${#REPLY} -gt $banner_cols ]] && banner_cols=${#REPLY}
    ((++banner_rows))
  done < <(banner "12:34 PM")
}

display_clock() {

  # Since we are putting the clock in the center of the terminal,
  # we need to read each line of banner's output and place it in the
  # right spot.

  local row=$clock_row

  while read; do
    tput cup $row $clock_col
    echo -n "$REPLY"
```

```
    ((++row))
  done < <(banner "$(date +'%I:%M %p')")
}

# Set a trap to restore terminal on Ctrl-c (exit).
# Reset character attributes, make cursor visible, and restore
# previous screen contents (if possible).

trap 'tput sgr0; tput cnorm; tput rmcup || clear; exit 0' SIGINT

# Save screen contents and make cursor invisible
tput smcup; tput civis

# Calculate sizes and positions
terminal_size
banner_size
clock_row=$(((terminal_rows - banner_rows) / 2))
clock_col=$(((terminal_cols - banner_cols) / 2))
progress_row=$((clock_row + banner_rows + 1))
progress_col=$(((terminal_cols - 60) / 2))

# In case the terminal cannot paint the screen with a background
# color (tmux has this problem), create a screen-size string of
# spaces so we can paint the screen the hard way.

blank_screen=
for ((i=0; i < (terminal_cols * terminal_rows); ++i)); do
  blank_screen="${blank_screen} "
done
```

```
# Set the foreground and background colors and go!
echo -n ${BG_BLUE}${FG_WHITE}
while true; do

  # Set the background and draw the clock

  if tput bce; then # Paint the screen the easy way if bce is supported
    clear
  else # Do it the hard way
    tput home
    echo -n "$blank_screen"
  fi
  tput cup $clock_row $clock_col
  display_clock

  # Draw a black progress bar then fill it in white
  tput cup $progress_row $progress_col
  echo -n ${FG_BLACK}
  echo -n "############################################################"
  tput cup $progress_row $progress_col
  echo -n ${FG_WHITE}

  # Advance the progress bar every second until a minute is used up
  for ((i = $(date +%S);i < 60; ++i)); do
    echo -n "#"
    sleep 1
  done
done
```

:

terminal clock

.

,

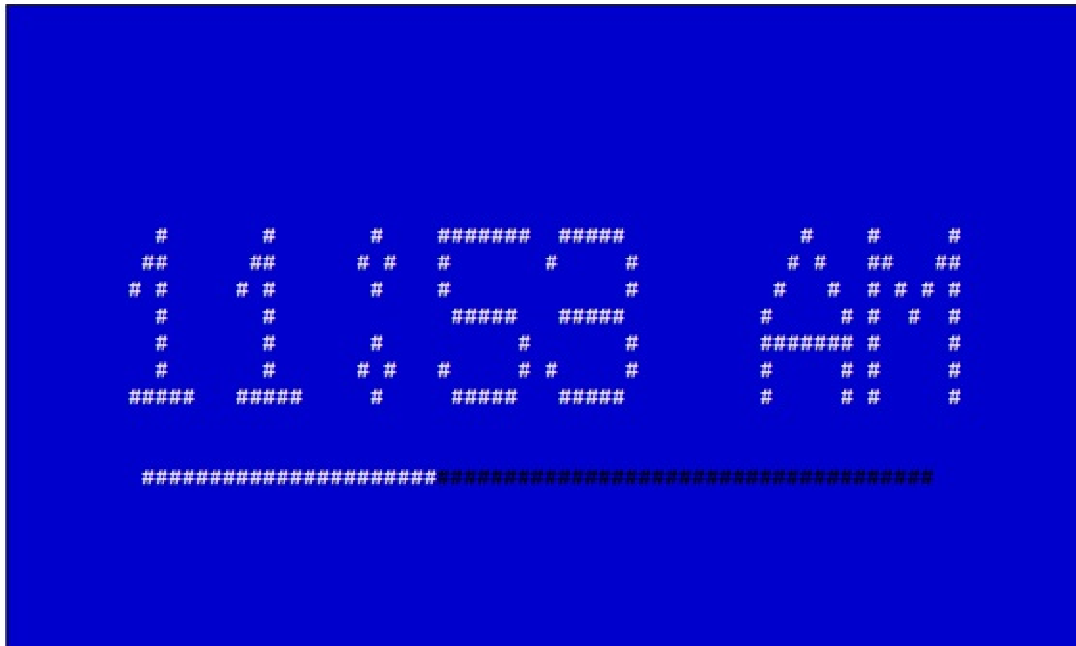
(progress bar)

.

(for loop) progress bar

.

:



□□□□□□□□□□□□□□:

– tput command

– script

( ) – clock

- output

– tensor

- progress bar

- terminal clock

– for loop

( )

– 30 (nohup command)

- நிரற்பா 30

:

, ,

(nohup)

.

(General syntax):

#nohup command-name &

#nohup /path/to/command-name arg1 arg2 &

.

#jobs –l

107

#niral107

#!/bin/bash

nohup find / -xdev -type f -perm +u=s -print > out.txt &

:

find

.

108

#niral108

#!/bin/bash

Example: Printing lines to both standard output & standard error

```
while(true)
do
echo "standard output"
echo "standard error" 1>&2
sleep 1;
done
```

1:

Execute the script without redirection

```
$ nohup sh custom-script.sh &
[1] 12034
$ nohup: ignoring input and appending output to `nohup.out'
$ tail -f nohup.out
standard output
standard error
standard output
standard error
..
```

2:

Execute the script with redirection

```
$ nohup sh custom-script.sh > custom-out.log &
[1] 11069
$ nohup: ignoring input and redirecting stderr to stdout
```

3:

$ tail -f custom-out.log
standard output
standard error
standard output
standard error
..

If you log-out of the shell and login again, you'll still see the custom-script.sh running in the background.



Term

- Need to run command in background with terminal detached?
- Use screen(1) & nohup(1)
  - screen - screen manager with VT100/ANSI terminal emulation
  - nohup - run a command immune to hangups, with output to a non-tty
  - Difference – while nohup is detaching command from terminal and redirecting output to file, screen is detaching itself (not command) from terminal.

4:

$ ps aux | grep prasanna
prasanna 12034 0.0 0.1 4912 1080 pts/2 S 14:10 0:00 sh custom-script.sh

nohup command with password less authentication:

(passwordless authentication)

,

(server)

.

,

.

$ nohup scp file_to_copy user@server:/path/to/copy/the/file > nohup.out 2>&1

,

,

nohup

.

:

– user

– background command

– terminal

– closed after the terminal

– give output

– nohup command

– passwordless authentication

- Server

( )

– 31

•

நிரற்பா 31

:

(General syntax):

109

```
#niral109
# Cleanup
# Run as root, of course.

cd /var/log
cat /dev/null > messages
cat /dev/null > wtmp
echo "Log files cleaned up."
```

:

(root user)

.

(permissions) .

110

```
#niral110
#!/bin/bash
# Proper header for a Bash script.

# Cleanup, version 2

# Run as root, of course.
# Insert code here to print error message and exit if not root.

LOG_DIR=/var/log
# Variables are better than hard-coded values.
cd $LOG_DIR

cat /dev/null > messages
cat /dev/null > wtmp
echo "Logs cleaned up."

exit #  The right and proper method of "exiting" from a script.
    #  A bare "exit" (no parameter) returns the exit status
    #+ of the preceding command.
```

:

.

(/var/log)

,

(reset) .

```
#niral111
#!/bin/bash
# Cleanup, version 3

#  Warning:
#  -------
#  This script uses quite a number of features that will be explained
#+ later on.
#  By the time you've finished the first half of the book,
#+ there should be nothing mysterious about it.

LOG_DIR=/var/log
ROOT_UID=0     # Only users with $UID 0 have root privileges.
LINES=50       # Default number of lines saved.
E_XCD=86       # Can't change directory?
E_NOTROOT=87   # Non-root exit error.

# Run as root, of course.
if [ "$UID" -ne "$ROOT_UID" ]
then
  echo "Must be root to run this script."
  exit $E_NOTROOT
fi

if [ -n "$1" ]
# Test whether command-line argument is present (non-empty).
then
  lines=$1
```

```
else
  lines=$LINES # Default, if not specified on command-line.
fi

#  Stephane Chazelas suggests the following,
#+ as a better way of checking command-line arguments,
#+ but this is still a bit advanced for this stage of the tutorial.
#
#    E_WRONGARGS=85  # Non-numerical argument (bad argument
format).
#
#    case "$1" in
#    ""      ) lines=50;;
#    *[!0-9]*) echo "Usage: `basename $0` lines-to-cleanup";
#     exit $E_WRONGARGS;;
#    *       ) lines=$1;;
#    esac
#
#* Skip ahead to "Loops" chapter to decipher all this.


cd $LOG_DIR

if [ `pwd` != "$LOG_DIR" ]  # or   if [ "$PWD" != "$LOG_DIR" ]
                            # Not in /var/log?
then
  echo "Can't change to $LOG_DIR."
  exit $E_XCD
fi  # Doublecheck if in right directory before messing with log file.
```

```
# Far more efficient is:
#
# cd /var/log || {
#   echo "Cannot change to necessary directory." >&2
#   exit $E_XCD;
# }

tail -n $lines messages > mesg.temp # Save last section of message log file.
mv mesg.temp messages               # Rename it as system log file.

#  cat /dev/null > messages
#* No longer needed, as the above method is safer.

cat /dev/null > wtmp  #  ': > wtmp' and '> wtmp'  have the same effect.
echo "Log files cleaned up."
#  Note that there are other log files in /var/log not affected
#+ by this script.

exit 0
#  A zero return value from the script upon exit indicates success
#+ to the shell.
```

:

,

.

.

:

– user

– new logs

– right manner

– clean

– root user

– reset

- settings

(   )

– 32

.

- நிரற்பா 32

:

,

, package

.

112

```
#!/bin/bash
#niral112
# rpm-check.sh

#  Queries an rpm file for description, listing,
#+ and whether it can be installed.
#  Saves output to a file.
#
#  This script illustrates using a code block.

SUCCESS=0
E_NOARGS=65
```

```
if [ -z "$1" ]
then
  echo "Usage: `basename $0` rpm-file"
  exit $E_NOARGS
fi

{ # Begin code block.
  echo
  echo "Archive Description:"
  rpm -qpi $1       # Query description.
  echo
  echo "Archive Listing:"
  rpm -qpl $1       # Query listing.
  echo
  rpm -i --test $1  # Query whether rpm file can be installed.
  if [ "$?" -eq $SUCCESS ]
  then
    echo "$1 can be installed."
  else
    echo "$1 cannot be installed."
  fi
  echo              # End code block.
} > "$1.test"       # Redirects output of everything in block to file.

echo "Results of rpm test in file $1.test"

# See rpm man page for explanation of options.

exit 0
```

:

, Archive

Description ,

Archive Listing

. SUCCESS

rpm can be installed

, cannot be

installed. .

.

.

113 - (tips script)

```
#!/bin/bash
#niral113

# uppercase.sh : Changes input to uppercase.

tr 'a-z' 'A-Z'
#  Letter ranges must be quoted
#+ to prevent filename generation from single-letter filenames.
Exit 0
```

:

.

. tr

translate

. uppercase.sh

. ls –l

.

ls -l | ./uppercase.sh

,

.

:

```
bash$ ls -l | ./uppercase.sh
-RW-RW-R--    1 BOZO  BOZO       109 APR  7 19:49 1.TXT
-RW-RW-R--    1 BOZO  BOZO       109 APR 14 16:48 2.TXT
-RW-R--R--    1 BOZO  BOZO       725 APR 20 20:56 DATA-FILE
```

114 (running a loop in background -

.)

```
#!/bin/bash
#niral114
# background-loop.sh

for i in 1 2 3 4 5 6 7 8 9 10            # First loop.
do
  echo -n "$i "
done & # Run this loop in background.
       # Will sometimes execute after second loop.

echo   # This 'echo' sometimes will not display.

for i in 11 12 13 14 15 16 17 18 19 20   # Second loop.
do
```

```
  echo -n "$i "
done

echo   # This 'echo' sometimes will not display.

# ==========================================================
```

:

. & ,
(loop)
(background) .
,
.
,
.
:

.

```
# The expected output from the script:
# 1 2 3 4 5 6 7 8 9 10
# 11 12 13 14 15 16 17 18 19 20

# Sometimes, though, you get:
# 11 12 13 14 15 16 17 18 19 20
# 1 2 3 4 5 6 7 8 9 10 bozo $
# (The second 'echo' doesn't execute. Why?)

# Occasionally also:
```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
(The first 'echo' doesn't execute. Why?)

Very rarely something like:
11 12 13 1 2 3 4 5 6 7 8 9 10 14 15 16 17 18 19 20
The foreground loop preempts the background one.
exit 0

:

– input
– output
– anlysing
– required
– non-required
- package
- script
– RPM package (Red Hat Package Manager)
– loop
- background
( )

– 33

.

- நிரற்பா 33

:

,

(shell script) (integer),
(string)

(run time)
(depends on the terminal)
(variable).

115

```
#!/bin/bash
#niral115.sh
# int-or-string.sh

a=2334                  # Integer.
let "a += 1"
echo "a = $a "          # a = 2335
echo                    # Integer, still.
```

```
b=${a/23/BB}            # Substitute "BB" for "23".
                        # This transforms $b into a string.
echo "b = $b"           # b = BB35
declare -i b            # Declaring it an integer doesn't help.
echo "b = $b"           # b = BB35

let "b += 1"            # BB35 + 1
echo "b = $b"           # b = 1
echo                    # Bash sets the "integer value" of a string to 0.

c=BB34
echo "c = $c"           # c = BB34
d=${c/BB/23}            # Substitute "23" for "BB".
                        # This makes $d an integer.
echo "d = $d"           # d = 2334
let "d += 1"            # 2334 + 1
echo "d = $d"           # d = 2335
echo


# What about null variables?
e=''                    # ... Or e="" ... Or e=
echo "e = $e"           # e =
let "e += 1"            # Arithmetic operations allowed on a null variable?
echo "e = $e"           # e = 1
echo                    # Null variable transformed into an integer.

# What about undeclared variables?
echo "f = $f"           # f =
```

```
let "f += 1"             # Arithmetic operations allowed?
echo "f = $f"            # f = 1
echo                     # Undeclared variable transformed into an integer.
#
# However ...
let "f /= $undecl_var"   # Divide by zero?
#   let: f /= : syntax error: operand expected (error token is " ")
# Syntax error! Variable $undecl_var is not set to zero here!
#
# But still ...
let "f /= 0"
#   let: f /= 0: division by 0 (error token is "0")
# Expected behavior.


#  Bash (usually) sets the "integer value" of null to zero
#+ when performing an arithmetic operation.
#  But, don't try this at home, folks!
#  It's undocumented and probably non-portable behavior.


# Conclusion: Variables in Bash are untyped,
#+ with all attendant consequences.

exit $?
```

:

untyped

,

,

.

,

. ,

(non-accurate scripts)

,

.

,

.

116:

```
#!/bin/bash
# niral116.sh

# Does a 'whois domain-name' lookup on any of 3 alternate servers:
#                    ripe.net, cw.net, radb.net

# Place this script -- renamed 'wh' -- in /usr/local/bin

# Requires symbolic links:
# ln -s /usr/local/bin/wh /usr/local/bin/wh-ripe
# ln -s /usr/local/bin/wh /usr/local/bin/wh-apnic
# ln -s /usr/local/bin/wh /usr/local/bin/wh-tucows

E_NOARGS=75



if [ -z "$1" ]
then
  echo "Usage: `basename $0` [domain-name]"
  exit $E_NOARGS
fi
```

```
# Check script name and call proper server.
case `basename $0` in    # Or:    case ${0##*/} in
    "wh"       ) whois $1@whois.tucows.com;;
    "wh-ripe"  ) whois $1@whois.ripe.net;;
    "wh-apnic" ) whois $1@whois.apnic.net;;
    "wh-cw"    ) whois $1@whois.cw.net;;
    *          ) echo "Usage: `basename $0` [domain-name]";;
esac

exit $?
```

:

,

,

.

. ripe.net, cw.net, radb.net

.

.

117:

,

.

.

echo

. escape sequences

. Linux

,

.

```
bash$ echo hello\!
hello!
bash$ echo "hello\!"
hello\!


bash$ echo \
>
bash$ echo "\"
>
bash$ echo \a
a
bash$ echo "\a"
\a

bash$ echo x\ty
xty
bash$ echo "x\ty"
x\ty

bash$ echo -e x\ty
xty
bash$ echo -e "x\ty"
x	y
```

:

- high level computer languages

- shellscript

- integer

- string

- variable

( )

– 34

(escape sequences)

.

- நிரற்பா 34

:

(script),

(escape sequence),

.

.

\n	means newline
\r	means return
\t	means tab
\v	means vertical tab
\b	means backspace

\a	means alert (beep or flash)
\0xx	translates to the octal ASCII equivalent of 0nn, where nn is a string of digits

118

```
#!/bin/bash
#niral118.sh
# escaped.sh: escaped characters
#############################################################
### First, let's show some basic escaped-character usage. ###
#############################################################
# Escaping a newline.
# ------------------
echo ""
echo "This will print
as two lines."
# This will print
# as two lines.
echo "This will print \
as one line."
# This will print as one line.
echo; echo
echo "============="
echo "\v\v\v\v"      # Prints \v\v\v\v literally.
# Use the -e option with 'echo' to print escaped characters.
echo "============="
echo "VERTICAL TABS"
echo -e "\v\v\v\v"   # Prints 4 vertical tabs.
echo "=============="
echo "QUOTATION MARK"
```

```
echo -e "\042"      # Prints " (quote, octal ASCII character 42).
echo "============="

# The $'\X' construct makes the -e option unnecessary.

echo; echo "NEWLINE and (maybe) BEEP"
echo $'\n'          # Newline.
echo $'\a'          # Alert (beep).
                    # May only flash, not beep, depending on terminal.

# We have seen $'\nnn" string expansion, and now . . .

#
==================================================================
#
# Version 2 of Bash introduced the $'\nnn' string expansion construct.
#
==================================================================
#

echo "Introducing the \$\' ... \' string-expansion construct . . . "
echo ". . . featuring more quotation marks."

echo $'\t \042 \t'   # Quote (") framed by tabs.
# Note that  '\nnn' is an octal value.

# It also works with hexadecimal values, in an $'\xhhh' construct.
echo $'\t \x22 \t'  # Quote (") framed by tabs.
# Thank you, Greg Keraunen, for pointing this out.
# Earlier Bash versions allowed '\x022'.
```

```
echo

# Assigning ASCII characters to a variable.
# ----------------------------------------
quote=$'\042'        # " assigned to a variable.
echo "$quote Quoted string $quote and this lies outside the quotes."

echo

# Concatenating ASCII chars in a variable.
triple_underline=$'\137\137\137'  # 137 is octal ASCII code for '_'.
echo "$triple_underline UNDERLINE $triple_underline"

echo

ABC=$'\101\102\103\010'           # 101, 102, 103 are octal A, B, C.
echo $ABC

echo
escape=$'\033'                    # 033 is octal for escape.
echo "\"escape\" echoes as $escape"
#                                   no visible output.
echo
exit 0
```

:

,

.

.

119:

```
#!/bin/bash
# niral119.sh
# Requires version 4.2+ of Bash.
key="no value yet"
while true; do
  clear
  echo "Bash Extra Keys Demo. Keys to try:"
  echo
  echo "* Insert, Delete, Home, End, Page_Up and Page_Down"
  echo "* The four arrow keys"
  echo "* Tab, enter, escape, and space key"
  echo "* The letter and number keys, etc."
  echo
  echo "    d = show date/time"
  echo "    q = quit"
  echo "================================"
  echo

 # Convert the separate home-key to home-key_num_7:
 if [ "$key" = $'\x1b\x4f\x48' ]; then
  key=$'\x1b\x5b\x31\x7e'
  #   Quoted string-expansion construct.
 fi

 # Convert the separate end-key to end-key_num_1.
 if [ "$key" = $'\x1b\x4f\x46' ]; then
  key=$'\x1b\x5b\x34\x7e'
 fi
```

```
case "$key" in
 $'\x1b\x5b\x32\x7e')  # Insert
  echo Insert Key
 ;;
 $'\x1b\x5b\x33\x7e')  # Delete
  echo Delete Key
 ;;
 $'\x1b\x5b\x31\x7e')  # Home_key_num_7
  echo Home Key
 ;;
 $'\x1b\x5b\x34\x7e')  # End_key_num_1
  echo End Key
 ;;
 $'\x1b\x5b\x35\x7e')  # Page_Up
  echo Page_Up
 ;;
 $'\x1b\x5b\x36\x7e')  # Page_Down
  echo Page_Down
 ;;
 $'\x1b\x5b\x41')  # Up_arrow
  echo Up arrow
 ;;
 $'\x1b\x5b\x42')  # Down_arrow
  echo Down arrow
 ;;
 $'\x1b\x5b\x43')  # Right_arrow
  echo Right arrow
 ;;
 $'\x1b\x5b\x44')  # Left_arrow
  echo Left arrow
```

```
 ;;
 $'\x09')  # Tab
  echo Tab Key
 ;;
 $'\x0a')  # Enter
  echo Enter Key
 ;;
 $'\x1b')  # Escape
  echo Escape Key
 ;;
 $'\x20')  # Space
  echo Space Key
 ;;
 d)
  date
 ;;
 q)
 echo Time to quit...
 echo
 exit 0
 ;;
 *)
  echo You pressed: \'"$key"\'
 ;;
esac
echo
echo "================================="
unset K1 K2 K3
read -s -N1 -p "Press a key: "
K1="$REPLY"
```

```
 read -s -N2 -t 0.001
 K2="$REPLY"
 read -s -N1 -t 0.001
 K3="$REPLY"
 key="$K1$K2$K3"
done
exit $?
```

:

.

:

– script

– escape sequence

– escape sequence

– assist

( )

– 35

(exit command)

.

- நிரற்பா 35

:

,

,

.

120

```
#!/bin/bash
#niral120.sh
echo hello
echo $?    # Exit status 0 returned because command executed successfully.

lskdf      # Unrecognized command.
echo $?    # Non-zero exit status returned -- command failed to execute.

echo

exit 113   # Will return 113 to shell.
```

To verify this, type "echo $?" after script terminates.

By convention, an 'exit 0' indicates success,
#+ while a non-zero exit value means an error or anomalous condition.

:

echo $? ,

. ,

,

.

.

121:

```
#!/bin/bash
# niral121.sh
true    # The "true" builtin.
echo "exit status of \"true\" = $?"     # 0

! true
echo "exit status of \"! true\" = $?"   # 1
# Note that the "!" needs a space between it and the command.
#    !true   leads to a "command not found" error
#
# The '!' operator prefixing a command invokes the Bash history mechanism.

true
!true
```

```
# No error this time, but no negation either.
# It just repeats the previous command (true).


#
==================================================================
#
# Preceding a _pipe_ with ! inverts the exit status returned.
ls | bogus_command     # bash: bogus_command: command not found
echo $?                # 127

! ls | bogus_command   # bash: bogus_command: command not found
echo $?                # 0
# Note that the ! does not change the execution of the pipe.
# Only the exit status changes.
#
==================================================================
#
```

:

, true, !true

. true

, !true

.

.

1	**Bash**	**let "var1 = 1/0"**
2	**.**	**empty_function() {}**
126	**.**	**/dev/null**
127		**illegal_command**
128	**.**	**exit 3.14159**
128+n		**kill -9 $PPID of script**
130	**ctrl+c .**	***Ctl-C***
255*	**.**	**exit -1**

Exit Status of a Script

- A script, like any other process, sets an exit status when it finishes executing. Shell scripts will finish in one of the following ways:

  `Abort` - If the script aborts due to an internal error, the exit status is that of the last command (the one that aborted the script).

  `End` - If the script runs to completion, the exit status is that of the last command in the script

  `Exit` - If the script encounters and exit command, the exit status is that set by that command

  - Syntax for the exit command is `exit (num)`. When the exit command is encountered the script ends right there and the exit status is set to `num`

OSC 71

□□□□□□□□□□□□□□□:

- commands

– without interruption

– more commands need to run

– exit command

– zero

– non-zero

– alternatively

– signal

– parameter

– fatal error

( )

– 36

(Internal Commands and Builtins)

.

- 36

□□□□□□□□□□□□□□□:

.

122

```
#!/bin/bash
#niral122.sh
#Internal Commands and Builtins
#A script that spawns multiple instances of itself
# spawn.sh

PIDS=$(pidof sh $0)  # Process IDs of the various instances of this script.
P_array=( $PIDS )    # Put them in an array (why?).
echo $PIDS           # Show process IDs of parent and child processes.
```

```
let "instances = ${#P_array[*]} - 1"  # Count elements, less 1.
                                # Why subtract 1?
echo "$instances instance(s) of this script running."
echo "[Hit Ctl-C to exit.]"; echo

sleep 1            # Wait.
sh $0              # Play it again, Prasanna.

exit 0             # Not necessary; script will never get to here.
                   # Why not?

#  After exiting with a Ctl-C,
#+ do all the spawned instances of the script die?
#  If so, why?

# Note:
# ----
# Be careful not to run this script too long.
# It will eventually eat up too many system resources.

#  Is having a script spawn multiple instances of itself
#+ an advisable scripting technique.
```

#Generally, a Bash builtin does not fork a subprocess when it executes within a script. An external system command or filter in a script usually will fork a subprocess.

:

,

. Ctrl+c

.

.

123:

```
#!/bin/bash
# niral123.sh

# Embedding a linefeed?
echo "Why doesn't this string \n split on two lines?"
# Doesn't split.

# Let's try something else.
echo

echo $"A line of text containing
a linefeed."
# Prints as two distinct lines (embedded linefeed).
# But, is the "$" variable prefix really necessary?

echo
echo "This string splits
on two lines."
# No, the "$" is not needed.
echo
echo "---------------"
echo

echo -n $"Another line of text containing
a linefeed."
# Prints as two distinct lines (embedded linefeed).
```

```
# Even the -n option fails to suppress the linefeed here.

echo
echo
echo "---------------"
echo
echo
# However, the following doesn't work as expected.
# Why not? Hint: Assignment to a variable.
string1=$"Yet another line of text containing
a linefeed (maybe)."

echo $string1
# Yet another line of text containing a linefeed (maybe).
#                                    ^
# Linefeed becomes a space.
```

:

.

.

124:

```
#!/bin/bash
#niral124.sh
# printf demo

declare -r PI=3.14159265358979     # Read-only variable, i.e., a constant.
declare -r DecimalConstant=31373
```

```
Message1="Greetings,"
Message2="Earthling."

echo

printf "Pi to 2 decimal places = %1.2f" $PI
echo
printf "Pi to 9 decimal places = %1.9f" $PI  # It even rounds off correctly.

printf "\n"                              # Prints a line feed,
                                         # Equivalent to 'echo' . . .

printf "Constant = \t%d\n" $DecimalConstant  # Inserts tab (\t).

printf "%s %s \n" $Message1 $Message2

echo

# ==========================================#
# Simulation of C function, sprintf().
# Loading a variable with a formatted string.

echo

Pi12=$(printf "%1.12f" $PI)
echo "Pi to 12 decimal places = $Pi12"      # Roundoff error!

Msg=`printf "%s %s \n" $Message1 $Message2`
echo $Msg; echo $Msg
```

```
#  As it happens, the 'sprintf' function can now be accessed
#+ as a loadable module to Bash,
#+ but this is not portable.

exit 0
```

Formatting error messages is a useful application of printf

```
E_BADDIR=85

var=nonexistent_directory

error()
{
  printf "$@" >&2
  # Formats positional params passed, and sends them to stderr.
  echo
  exit $E_BADDIR
}

cd $var || error $"Can't cd to %s." "$var"
```

: (printf command in shell script)

,

. printf

,

,

.

:

- default

- - reason of

- scripts

- output

- embedded

- text

- built-in

- escape sequences

( )

– 37

.

- 37

:

.

124:

```
#!/bin/bash
# arith-tests.sh
# Arithmetic tests.

# The (( ... )) construct evaluates and tests numerical expressions.
# Exit status opposite from [ ... ] construct!

(( 0 ))
echo "Exit status of \"(( 0 ))\" is $?."         # 1

(( 1 ))
echo "Exit status of \"(( 1 ))\" is $?."         # 0
```

```
(( 5 > 4 ))                      # true
echo "Exit status of \"(( 5 > 4 ))\" is $?."     # 0

(( 5 > 9 ))                      # false
echo "Exit status of \"(( 5 > 9 ))\" is $?."     # 1

(( 5 == 5 ))                     # true
echo "Exit status of \"(( 5 == 5 ))\" is $?."    # 0
# (( 5 = 5 ))  gives an error message.

(( 5 - 5 ))                      # 0
echo "Exit status of \"(( 5 - 5 ))\" is $?."     # 1

(( 5 / 4 ))                      # Division o.k.
echo "Exit status of \"(( 5 / 4 ))\" is $?."     # 0

(( 1 / 2 ))                      # Division result < 1.
echo "Exit status of \"(( 1 / 2 ))\" is $?."     # Rounded off to 0.
                                 # 1

(( 1 / 0 )) 2>/dev/null                    # Illegal division by 0.
#           ^^^^^^^^^^^
echo "Exit status of \"(( 1 / 0 ))\" is $?."     # 1

# What effect does the "2>/dev/null" have?
# What would happen if it were removed?
# Try removing it, then rerunning the script.

# ======================================= #
```

```
# (( ... )) also useful in an if-then test.

var1=5
var2=4

if (( var1 > var2 ))
then #^      ^      Note: Not $var1, $var2. Why?
  echo "$var1 is greater than $var2"
fi     # 5 is greater than 4

exit 0
```

:

,

. ,

.

1. நிகராக இருப்பது (equals)
2. அதிகமாக இருப்பது (more than)
3. குறைவாக இருப்பது (less than)
4. அதிகமாக நிகராக இருப்பது (more than or equal)
5. குறைவாக நிகராக இருப்பது (less than or equal)

.

125:

```
#!/bin/bash
# broken-link.sh
# Written by PNA Prasanna
# Used in ABS Guide with permission.
```

```
#  A pure shell script to find dead symlinks and output them quoted
#+ so they can be fed to xargs and dealt with :)
#+ eg. sh broken-link.sh /somedir /someotherdir|xargs rm
#
#  This, however, is a better method:
#
#  find "somedir" -type l -print0|\
#  xargs -r0 file|\
#  grep "broken symbolic"|
#  sed -e 's/^\|: *broken symbolic.*$/"/g'
#
#+ but that wouldn't be pure Bash, now would it.
#  Caution: beware the /proc file system and any circular links!
##################################################################


#  If no args are passed to the script set directories-to-search
#+ to current directory.  Otherwise set the directories-to-search
#+ to the args passed.
######################

[ $# -eq 0 ] && directorys=`pwd` || directorys=$@


#  Setup the function linkchk to check the directory it is passed
#+ for files that are links and don't exist, then print them quoted.
#  If one of the elements in the directory is a subdirectory then
#+ send that subdirectory to the linkcheck function.
##########
```

```
linkchk () {
   for element in $1/*; do
     [ -h "$element" -a ! -e "$element" ] && echo \"$element\"
     [ -d "$element" ] && linkchk $element
   # Of course, '-h' tests for symbolic link, '-d' for directory.
   done
}

#  Send each arg that was passed to the script to the linkchk() function
#+ if it is a valid directoy.  If not, then print the error message
#+ and usage info.
##################
for directory in $directorys; do
   if [ -d $directory ]
then linkchk $directory
else
   echo "$directory is not a directory"
   echo "Usage: $0 dir1 dir2 ..."
   fi
done

exit $?
```

:

– arithmetic operations

– double brackets

- logic

( )

:



.

.

.

, “ ”

,

.

,

,

.

“ ”

!!

- 38

:



...

. . . . .

. .

,

.

. .

.

2008

, ,

.